NELC / TR 1778

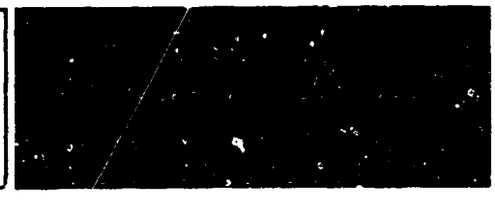
MATHEMATICAL PROGRAMMING AS AN AID TO ENGINEERING DESIGN

A user's guide to available MP computer codes and to NELC's capabilities in numerically solving MP problems

D. C. McCall

Research and Development

11 August 1971



NAVAL ELECTRONICS LABORATORY CENTER
SAN DIEGO, CALIFORNIA 92152

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Naval Electronics Laboratory Center San Diego, California 92152	UNCLASSIFIED
	2b. GROUP

3. REPORT TITLE

MATHEMATICAL PROGRAMMING AS AN AID TO ENGINEERING DESIGN

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Research and Development July 1968 to March 1970

5. AUTHOR(S) (First name, middle initial, last name)

D. C. McCall

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
11 August 1971	132	66

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO. ZFXX.212.001 (NELC Z223)	1778
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Director of Navy Laboratories

13. ABSTRACT

This report is a user's guide to available mathematical programming (MP) computer codes and to NELC's capabilities in numerically solving MP problems. It defines the general MP problem; lists and evaluates the MP codes operational on the NELC IBM 360/65 computers; and provides guidelines for modifying the MP problem when, in its first form, it is cumbersome; or there is not information enough to start computation; or the available codes do not yield all the needed information.

DD FORM 1 NOV 65 1473 (PAGE 1)

0102-014-6600

UNCLASSIFIED

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer programming						
Mathematical programming						

PROBLEM

Provide analysis and synthesis of Navy design problems. Specifically, develop a capability at NELC for using mathematical programming (MP) as an aid to engineering design.

RESULTS

1. The overall software capabilities at NELC for numerically solving various types of MP problems – initiated or developed under this problem – are discussed in the report proper. Applications of integer programming are given in Appendix 1. User's guides and FORTRAN codes for solving some classes of MP problems are given in Appendix 2.

2. Practical guidelines are given for applying MP methods.

RECOMMENDATIONS

1. Maintain a continued effort to keep the mathematical programming software current. Monitor research literature for new developments in non-linear programming and integer programming.

2. Conduct ongoing seminars or in-house classes to inform practicing scientists and engineers of the utility of MP.

3. Review Navy engineering design problems for possible application of MP techniques.

ADMINISTRATIVE INFORMATION

Work was performed under ZFXX.212.001 (NELC Z223) by the Decision and Control Technology Division. The report covers work done from July 1968 to March 1970 and was approved for publication 11 August 1971.

The author thanks W. J. Dejka, Advanced Modular Concepts Division, who, as the initial principal investigator of Z223, set the goals and general trends of the project, for assistance and comments on this report; and San Diego State College Foundation students, Gail Grotke and Dale Klamer, and C. M. Becker, Applications Software Division, for programming assistance in preparing Appendix 2.

CONTENTS

ILLUSTRATIONS

TABLE

INTRODUCTION: SCOPE OF REPORT

There are several problems involved in the development of faithful mathematical models of real-world processes. The processes are in general nonlinear. The modeling equations are frequently incomplete. Conditions are known only within limits. Often the best approach to these problems is via mathematical programming (MP).[1-5]

MP is a distinct discipline – it exists independently of computer programming. Prior to the age of the high-speed computer, however, some of the original algorithms for the solution of MP problems were too cumbersome to be of real use. The advent of the modern digital computer has made the solution of many types of MP problems feasible and has stimulated the search for better algorithms.

This report is chiefly concerned with solving MP problems – with the computational stage of MP. It is intended as a guide for the usage and application of available MP computer codes.

THE MATHEMATICAL PROGRAMMING PROBLEM defines the general MP problem.

MATHEMATICAL PROGRAMMING CAPABILITIES AT NELC lists and evaluates the MP codes operational on the NELC IBM 360/65 computer, and will be of interest to the user who has an MP problem in final form, ready to solve. He can choose the appropriate code from the list and obtain the card deck and user's guide from the NELC program library, Computer Sciences Department.

DESIGNING WITH MATHEMATICAL PROGRAMMING AS AN AID provides guidelines for modifying the MP problem when, in its first form, it is cumbersome; or when there is not information enough to start computation (for example, an initial feasible point is lacking); or when the available codes do not yield all the needed information (for example, postoptimal analysis).

---

1. See APPENDIX 3: REFERENCES.

THE MATHEMATICAL PROGRAMMING PROBLEM

The basic problem of MP is to develop an algorithm for finding the minimum of a scalar-valued function of n real variables that satisfies a set of auxiliary conditions called constraints. Stated in mathematical terms, the problem becomes:

Let $f(x_1, \ldots, x_n)$ and $g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)$ be scalar-valued functions of the n variables $x_1, \ldots, x_n$. Then we wish to find variables which

$$\text{minimize } f(x_1, x_2, \ldots, x_n)$$

subject to

$$\begin{aligned} g_1(x_1, x_2, \ldots, x_n) &\geq 0 \\ &\vdots \\ g_m(x_1, x_2, \ldots, x_n) &\geq 0 \end{aligned} \tag{1}$$

The above problem is known variously as the 'general mathematical programming problem,' the 'constrained optimization problem,' and the 'nonlinear programming problem.' For the sake of convenience, we call it problem (1).

In problem (1), f is called the objective or cost function and the g_i are called the constraints. We also refer to f and g_i as the problem functions. We denote the vector $(x_1, x_2, \ldots, x_n)$ by x^T (where T denotes the matrix transpose) and call the set of all vectors x which satisfy $g_i(x) \geq 0$ for all $i=1, \ldots, m$, the constraint set or feasible set. The problem is said to be consistently posed if the constraint set is nonempty. We note that finding the maximum of a function f is equivalent to finding the minimum of -f.

Consider the following example (fig. 1):

$$\text{Minimize} \quad f(x_1, x_2) = (x_1-2)^2 + (x_2-1)^2$$

subject to

$$\begin{aligned} g_1(x_1, x_2) &= x_2 - x_1^2 \geq 0 \\ g_2(x_1, x_2) &= -x_1 - x_2 + 2 \geq 0 \end{aligned} \tag{2}$$

Since $f(x_1, x_2)$ is the sum of squares, the minimum occurs at $x_1 = 2$, $x_2 = 1$. However, the point (2,1) is not in the constraint set defined by g_1 and g_2. The obvious (in this example) candidate is the point (1,1), which in this straightforward problem is the constrained minimum. As m and n get larger, the problem becomes significantly more difficult to solve.

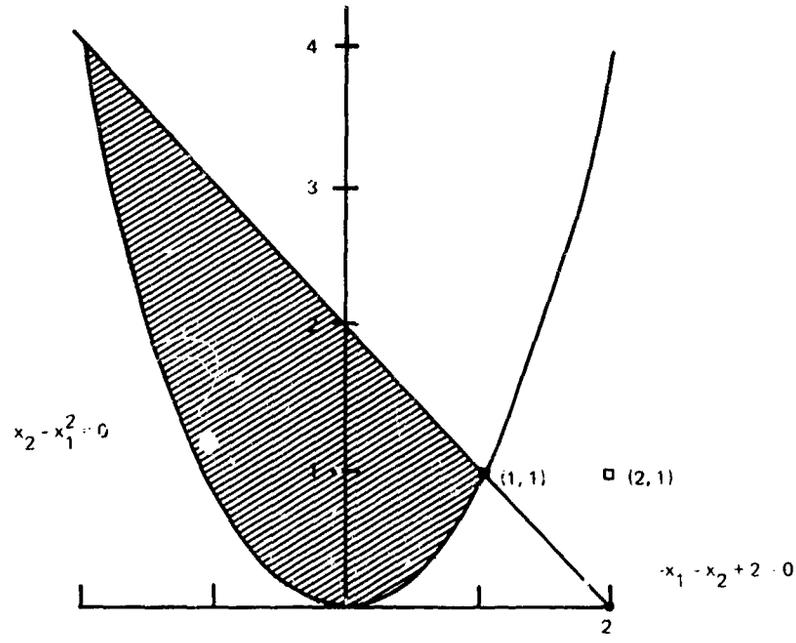


Figure 1. Constraint set: MP problem.

We now turn to a discussion of computer codes which can solve MP problems, for different classes of problem functions.

MATHEMATICAL PROGRAMMING CAPABILITIES AT NELC

UNCONSTRAINED PROBLEMS

The unconstrained MP problem is stated as:

Minimize $f(x_1, x_2, \ldots, x_n)$

where f is a scalar-valued function of the n variables $(x_1, x_2, \ldots, x_n)^T = \mathbf{x}$.

Gradient methods and direct search techniques are the two basic† approaches to numerically solving the unconstrained problem. The gradient of f at $\mathbf{x}$, denoted by $\nabla f(\mathbf{x})$, is defined to be the following vector

$$\nabla f(\mathbf{x}) = (\partial f(\mathbf{x})/\partial x_1, \ldots, \partial f(\mathbf{x})/\partial x_n)^T$$

Gradient methods use, in some way, the following facts:

1. At the minimum $\mathbf{x}^*$ of f, we have $\nabla f(\mathbf{x}^*) = 0$.
2. If $\nabla f(\mathbf{x}) \neq 0$, then $-\nabla f(\mathbf{x})$ points in the direction of steepest descent.

†A finer classification would be direct methods, gradient methods, and those methods involving the matrix of second derivatives. We feel that the first two are the most useful for applications.

This analytic information makes the gradient methods fast and enables the computer codes to compute meaningful error information. To use gradient methods, we must have the gradient of f available analytically or have a numerical way to compute it. Direct search methods eliminate this need for the gradient and rely only on the behavior of the objective function in seeking out the minimum. Typically, direct methods evaluate the cost function many more times than gradient methods in minimizing the same test function. In minimizing the Rosenbrock test function (see Appendix 2), the direct search routine ZANGWI requires 325 function evaluations, while the gradient method CONJGT requires only 71 combined function and gradient evaluations, in finding the optimum to within the same accuracy. This is a trade-off a user must make if he can choose between a gradient method and a direct method.

Direct methods do not rest on so firm a mathematical foundation as gradient methods do, and most direct methods are proved to converge for only special functions. However, they have been useful in practice, since the objective function in many applications is complicated or its gradient is not available. It is generally simpler to code a problem for a direct method, which allows for faster implementation on the computer.

We present the following example of posing a two-point boundary value problem (TPBVP) as an unconstrained MP problem, to illustrate both an application of MP and the need for good direct search methods. The problem is to find an n-dimensional vector function $\mathbf{y}(t)$ which satisfies

$$\dot{\mathbf{y}} = \mathbf{h}(t,\mathbf{y}) \qquad (a \leqslant t \leqslant b)$$

with

$$y_i(a) = q_{ia} \qquad (i=1,2,\ldots,j<n)$$

and

$$y_i(b) = q_{ib} \qquad (i=j+1,\ldots,n)$$

In general, this problem has no closed-form solution, and in some cases no solution at all. However, since it frequently arises in applications, either a numerical estimate of the solution is desired or, if no solution exists, a function which comes close (in some sense) to solving the problem is desired. With this in mind we pose the following MP problem.†

$$\text{Minimize } f(x_1, x_2, \ldots, x_{n-j}) = \left(\sum_{i=j+1}^{n} (y_i(b)-q_{ib})^2 \right)^{1/2} \qquad (3)$$

---

†Rosen[6] discusses the same problem and obtains approximate solutions using linear programming techniques. His approach requires a great deal of equation manipulation before the linear programming techniques can be applied. Unfortunately, no comparison between the two methods has been made.

where the numbers $y_i(b)$, $i=j+1, \ldots, n$ are numerically computed as follows. For a given $(x_1, \ldots, x_{n-j})$, solve the following initial-value problem over the interval [a,b].

$$\dot{y} = h(t,y) \qquad (a \leqslant t \leqslant b)$$

where

$$y(a) = (q_{1a}, \ldots, q_{ja}, x_1, \ldots, x_{n-j})$$

the last n-j components of the solution obtained at t=b of this problem are used in the objective function for $y_i(b)$, $i=j+1, \ldots, n$.

In this problem there is no analytic expression for the objective function f, from which ∇f can be derived. Thus, to numerically solve this unconstrained problem (3), either a direct search method must be used or ∇f must be numerically calculated via a differencing routine. We recommend the former as reliable and easy to use, and discuss some of the drawbacks of the latter in DESIGNING WITH MATHEMATICAL PROGRAMMING AS AN AID, Gradient approximation. Dejka[7] discusses a similar TPBVP and uses a direct search method to solve a related MP problem.

We briefly discuss computer routines available from the NELC program library for solving the unconstrained problem. The user's guides for these routines provide ample background information and details for implementation.

Gradient methods from the library are FP, CONJGT, SOREN, FMFP, and FMCG. FP and CNJGAT originally were programmed and used by Winterbauer[8] to solve a parameter selection problem for a sonar signal equation, but they are general-purpose, unconstrained, MP codes. FP and CNJGAT are based on the methods of Fletcher and Powell[9] and Fletcher and Reeves,[10] respectively. SOREN is a modification of CNJGAT which has converged faster for some test functions. FMFP and FMCG are available from the IBM Scientific Subroutine Package;[11] we have not tested these last two routines extensively or compared them with the other gradient methods.

ZANGWL, DIRECT, and UNIVAR are direct search codes in the library. ZANGWL is discussed in Appendix 2, and DIRECT and UNIVAR are presented in reference 12. These three routines are based on methods presented in references 13-15, respectively, and were programmed at NELC. ZANGWL has a mathematical basis for convergence similar to that of CNJGAT, and of the three direct methods, it is the most efficient in terms of the total number of function evaluations required to minimize a function. In minimizing the Rosenbrock function, to the same accuracy and from the same initial point, the number of function evaluations were ZANGWL(325), DIRECT(705), and UNIVAR(2303). The extraordinary number of function evaluations clearly makes UNIVAR unacceptable, but ZANGWL should not be selected over DIRECT. DIRECT makes intermediate searching moves in a much more

cautious manner than ZANGWL, which makes it better for some applications. This is discussed in DESIGNING WITH MATHEMATICAL PROGRAMMING AS AN AID, SUMT and Constraint Transformation.

Before moving onto the constrained problem, we give a word of caution on all minimization routines. Each method, be it a constrained or an unconstrained code, is capable of finding only a local minimum and not a global minimum. We define local and global minima in the next lines. Let the objective function f(x) be defined on a set G in an n-dimensional vector space, denoted by E^n. Then we say that f has a global minimum at x^* (in G) if $f(x^*) \leqslant f(x)$ for all x in G. Note that we do not exclude $G \equiv E^n$. $\hat{x}$ in G is called a local minimum, if for all x sufficiently close (with respect to some norm) to $\hat{x}$, and also in G, we have $f(\hat{x}) \leqslant f(x)$.

Local minima can occur in the gradient methods because the condition that $\nabla f(x^*) = 0$ is only necessary and not sufficient for a global minimum. In direct methods, only local information about the surface defined by the objective function is available to the routine. This characteristic makes direct methods susceptible to stopping at a local minimum. For reasonable certainty that a global optimum has been reached, it is wise to restart the problem from different initial points. In many applications, if a local minimum gives a satisfactory value of the objective function, no further processing is necessary.

Work is continuing in the area of unconstrained minimization algorithms, with refinements to the above methods and new methods appearing regularly in the literature. The most fruitful and accessible sources of articles on the subject are The Computer Journal, Communications of the Association of Computing Machinery, SIAM Review, and the SIAM journals on control, numerical analysis, and applied mathematics. The above sources, together with Management Science and Operations Research, contain many articles on the constrained problem.

CONSTRAINED PROBLEMS

We return to the discussion of problem (1) for various classes of problem functions. The following types of mathematical programming problems are discussed: linear, quadratic, convex, nonlinear and nonconvex, and integer. The methods for solving these problems make explicit use of the properties of the problem functions.

When an unconstrained problem is solved, the codes require only an initial point for which the objective function is defined. This requirement is more demanding in the constrained problem. Depending on the type of problem under consideration, the user can be required to provide an initial feasible point, as a starting point for the computation. In many applications an initial feasible point is known from the engineering knowledge of the problem. However, if the constraints are numerous or complicated, such a point will not be

obvious and a preliminary step must be taken prior to solving the problem. A method for obtaining an initial feasible point is treated in DESIGNING WITH MATHEMATICAL PROGRAMMING AS AN AID, Initial Points and Scaling, so we assume that one is at hand in the following discussion.

Each class of constrained MP problem is described, together with computer programs which can solve it. Since each routine to be discussed has an associated user's guide, we confine our remarks to the following points:

1. Is an initial feasible point required?
2. Does the code find a global optimum?
3. Can information be saved for possible restarts or postoptimal analysis?
4. Is the routine easy to use?
5. What error messages are given if the routine fails to converge?

LINEAR PROGRAMMING (LP)

In the standard linear programming problem all the functions in question are linear and the problem variables are constrained to be non-negative. We write:

$$\text{Minimize } f(\mathbf{x}) = c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

subject to

$$g_1(\mathbf{x}) = a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leqslant b_1$$

$$\vdots$$

$$g_m(\mathbf{x}) = a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leqslant b_m$$

This LP problem can be written in matrix notation as:

$$\text{Minimize } z = \mathbf{c}^T\mathbf{x}$$

subject to

$$\mathbf{Ax} \leqslant \mathbf{b}$$

$$x_i \geqslant 0$$

where the $\leqslant$ means that the corresponding components of the vectors are "less than or equal to." If the constraints are consistent, then the simplex method of linear programming guarantees that a global optimum† can be found in a finite number of steps. The simplex method is an iterative procedure and generates "basic feasible solutions" at each iteration, which decrease z. To produce these solutions, a "basis inverse" matrix is calculated. The

†An unbounded solution can also be detected in a finite number of steps.

preceding brief comments serve only to associate the terms "basic feasible solution" and "basis inverse" with the simplex method; references 16 and 17 treat the simplex method.

The most complete code for numerically solving the LP problem is the IBM Mathematical Programming System[18] (MPS/360). MPS/360 is based on a modification of the simplex method and will either solve the LP problem or indicate that no solution exists. This code does not require that the problem variables be nonnegative, and treats upper and lower bounds on the variables as special constraints. Separable programming problems (a special nonlinear MP problem) can be solved with this routine. MPS/360 is capable of solving problems of up to 4095 constraints and "virtually an unlimited number of columns."[18] It is currently stored on disk pack NELC05 at the NELC Computer Center.

No initial point is required to begin the computation; however, the option exists to start the problem from a user-supplied basis inverse. MPS/360 has its own control language, which provides a variety of capabilities. The user is afforded several postoptimal analysis procedures and can access the current basis inverse for future restarts. This control language is straightforward to use and provides some looping and branching capability. A variety of messages are output to the user in the course of computation. They are fully explained in the message manual.[19]

The chief drawback of this program is the format of the input data. It requires each element of the arrays c,b, and A to have a "row name" and a "column name" for identification. This has proved cumbersome for scientific and engineering work. A FORTRAN program, DATAPREP, is available to reduce the data arranged in compact matrix notation to a format acceptable to MPS/360.

In many applications a linear programming problem must be solved repeatedly as part of a larger problem. The READCOMM[20] facility of MPS/360 allows the main program to be used in an iterative fashion as a subroutine. READCOMM enables the user to supplement the standard control language with FORTRAN procedures; for example, DATA PREP. Rosen[6] and Griffith and Stewart[21] have examples of using a linear programming code in an iterative way.

Previous large-scale, efficient LP codes were geared to commercial applications and required a great deal of modification for efficient scientific and engineering use. The READCOMM facility has made a powerful program easily available for a wide range of specialized applications.

QUADRATIC PROGRAMMING (QP)

This type of problem is the next order of difficulty. A quadratic cost function is minimized subject to linear constraints:

Minimize $f(x_1, x_2, \ldots, x_n)$

subject to

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &\leq b_1 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &\leq b_m \end{aligned} \tag{4}$$

where f has one of two forms –

$$f(x_1, x_2, \ldots, x_n) = c^T x + x^T B x \tag{5}$$

or

$$f(x_1, x_2, \ldots, x_n) = \|Hx - e\| \tag{6}$$

B and H are n-by-n and k-by-n matrices, respectively, and **c** and **e** are n-, and k-dimensional vectors, respectively. The norm of a vector **y**, denoted by $\|y\|$, is given by

$$\|y\| = \left(\sum_{i=1}^{n} y_i^2 \right)^{1/2}.$$

At present only the minimum norm problem (equation 6) can be solved at NELC. The program which does this is QPHANSON. This routine was written by R. J. Hanson, of the Jet Propulsion Laboratory, Pasadena, and uses a numerically stable† version of Rosen's[22] gradient projection algorithm. The method guarantees that a global optimum will be found for a consistent problem. The routine is reported to have worked well on examples from the areas of curve fitting and approximation of solutions to linear integral equations.[23]

The routine is described in reference 23, and an NELC user's guide is in preparation. The code is operational, but it is still unpolished in respect to user-oriented input and output. QPHANSON does not require an initial feasible point nor does it have an option to accept a good approximation of the optimum. The code treats equality constraints and inequality constraints separately. It can solve problems of up to 60 constraints (equality and inequality combined) and 30 variables. No experiments have been done to determine whether this is a hard and fast upper bound on the problem size.

The program suffers from lack of good error and timing messages. If the routine fails to converge, no messages are given as to the possible cause. Also, no provisions are made to identify inconsistent quadratic programs. The user is on his own with QPHANSON.

---

†An algorithm ϕ is numerically stable if the errors in the input data are approximately equal to the round-off errors generated by the computations of ϕ.

If a QP problem occurs with equation (5) as the cost function, and the matrix B is positive-definite or positive-semidefinite,[††] then Hanson[23] presents a method for transforming this QP problem into a minimum-norm QP problem. This minimum-norm problem is solved with QPHANSON and then an inverse transformation is made. Presently this must be done by the user. QPHANSON is currently being modified to make this transformation automatically.

Codes to solve quadratic programs are not as well polished or as highly developed as those for LP. Unless the demand increases for good QP routines, the user will have to write his own code or be content with the experimental models.

CONVEX PROGRAMMING

Before turning to the convex programming problem, we make some preliminary definitions.

CONVEX SET. A set G in E^n is said to be convex (fig. 2) if for any two points x_1 and x_2 contained in G we have $\lambda x_1 + (1-\lambda)x_2$ contained in G, for all λ in (0,1).

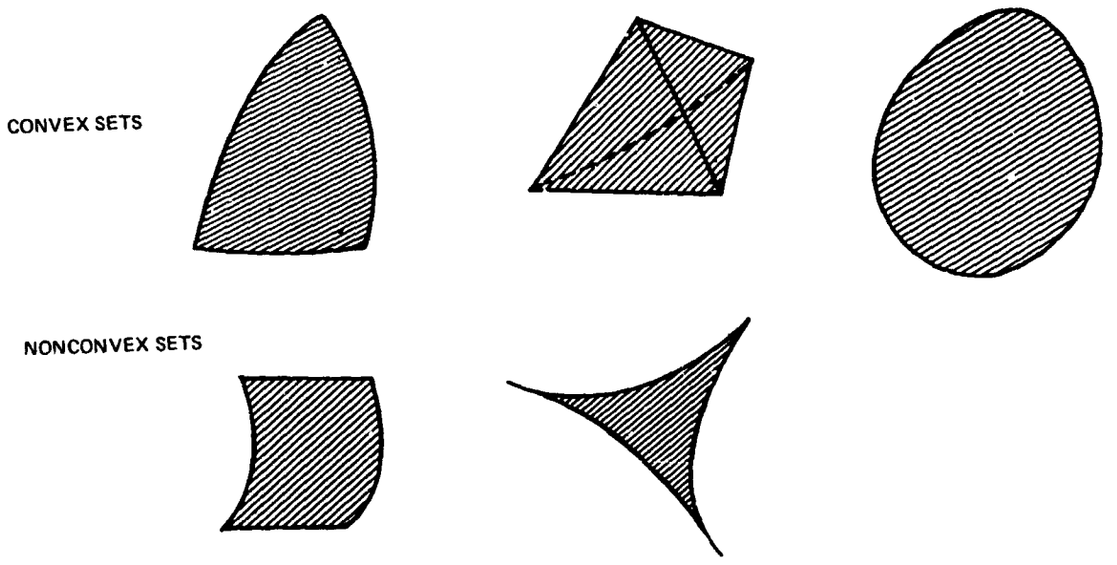


Figure 2. Examples of convex and nonconvex sets.

[††] A symmetric matrix B is positive-definite (semidefinite) if for every $x \neq 0$ we have $x^T Bx > 0, (\geq 0)$.

CONVEX FUNCTION. A scalar-valued function f defined on a convex set G in E^n is said to be convex if for any two points $\mathbf{x}_1$ and $\mathbf{x}_2$ in G

$$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leqslant \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)$$

for all λ in (0,1).

Linear functions, and the quadratic cost function (equation 6) with B positive-semidefinite, are examples of convex functions. A theorem of interest states that if the constraint set of an MP problem is defined by convex functions, then it, too, is convex. More precisely, if $g_1(\mathbf{x}), \ldots, g_m(\mathbf{x})$ are convex functions, then the set of all $\mathbf{x}$ for which $g_1(\mathbf{x}) \leqslant 0, \ldots, g_m(\mathbf{x}) \leqslant 0$ holds simultaneously is a convex set. The constraint sets of linear and quadratic programs are convex.

With these facts in hand we state the convex programming problem.

Minimize $f(\mathbf{x})$

subject to

$$g_i(x) \leqslant 0 \qquad (i=1, \ldots, m)$$

where the functions f and g_i are convex functions of $\mathbf{x} = (x_1, \ldots, x_n)^T$.

A great deal of work has been done with convex programming and the theory[24] can guarantee convergence for some computational methods. Each method is valid for specific requirements on the problem functions. In this section we assume that the gradients of all functions exist and are continuous. The central problem in solving convex programs is not so much theoretical difficulty but rather the obtaining of rapid convergence of numerical schemes. Even though some QP problems are convex, it may be more efficient to use a routine like QPHANSON to solve them rather than treat them as convex programs. Another practical difficulty with convex programming is the identification of convex functions. If the function has a complicated analytic expression, it can be difficult to classify it as convex. The methods for solving convex programs will not in general completely hang-up if the data are not convex, but the significance of such results should be judged in terms of the user's problem formulation.

The first library routine which solves the convex programming problem is the subroutine CONVEX, which was developed by Hartley and Hocking[25] at Texas A&M. The routine makes a linear approximation to the functions in question and then uses a simplex-like procedure to move to the optimum. In making the linear approximations, the routine requires a user-supplied subroutine which computes the gradients of the cost function and the nonlinear constraints.

CONVEX does not require an initial feasible point; however, the option does exist to start from a given point. In addition, CONVEX produces

a current feasible point and a basis inverse at the end of each iteration for possible restarts. The format of the input data is straightforward and suitable to scientific and engineering work; however, care should be exercised in the organization of the data for any upper and lower bounds on the problem variables. CONVEX requires that the constraint data be input in three groups – the upper and lower bounds on the variables, the linear inequalities, and the nonlinear convex constraints. This feature makes it possible to conveniently solve quadratic programs with convex cost functions. No comparisons between QPHANSON and CONVEX have been made on solving quadratic programming problems.

CONVEX suffers from the lack of good error messages and analysis in the event of an inconsistent problem or any numerical difficulties. No investigation of the numerical stability of the method or of timing or accuracy benchmarks for large problems has been reported. A convex problem with 60 constraints and 60 variables is the largest which can be solved without program modifications. Because the linear constraints are treated separately, it is likely that larger problems can be solved if the number of nonlinear constraints is not too great. Future work should investigate this possibility.

The second routine for solving the convex programming problem is Experimental SUMT. This method is theoretically convergent for convex data, but, since it also has provisions for nonconvex programs, we postpone discussion of it until the next section.

There is a special subclass of convex programs for which a global optimum can be found with the linear programming code MPS/360. These are separable programming problems, which are defined as follows:

$$\text{Minimize } z = \sum_{j=1}^{n} f_j(x_j) \tag{7}$$

subject to

$$\sum_{j=1}^{n} g_{ij}(x_j) \leq b_i \qquad (i=1, \ldots, m)$$

Note that the objective function and the constraints are sums of functions of the single variables x_j; that is, there are no "cross product" terms. This allows each nonlinear function to be replaced by a polygonal approximation, and reduces problem (7) to a form which can be solved by MPS/360. The MPS/360 user's manual[18] gives the appropriate details and examples of solving separable programming problems. If the separable functions are convex, then MPS/360 will find a global optimum to the approximation problem. The use of successive approximations causes the global optima of the approximation problems to converge to the optimum of the original

problem. The method can tolerate some nonconvex functions but may stop at a local optimum.

This technique can also be used for separable convex programs too large for CONVEX. (CONVEX may be more efficient if the problem is not too large – unfortunately, no experimental evidence is available to aid in making the selection.)

NONLINEAR, NONCONVEX PROGRAMMING

This last class of MP problems is composed of all the problems which are not necessarily linear or necessarily convex. In the statement of problem (1), no requirements were made on the functions in question other than the assumptions that the objective function would be defined for all feasible $\mathbf{x}$, and the g_i would be defined for all $\mathbf{x}$. This statement of problem (1) is much too general to be of use. To have any hope of obtaining a solution, we must put some restrictions on the problem functions. The three computer codes which we discuss require that the gradients of all the functions in problem (1) exist and be continuous. Although these conditions are stringent from a mathematical point of view,† they do not provide a base for an MP algorithm.

The following example illustrates one of the difficulties of nonlinear, nonconvex programming. Since the problem functions are possibly nonconvex, complicated constraint sets can be generated (fig. 3).

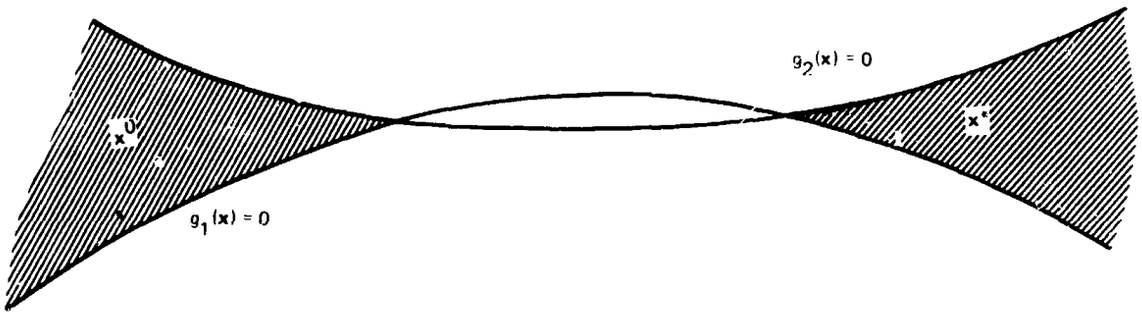


Figure 3. Disconnected constraint set.

In this example, if the initial feasible point $\mathbf{x}^0$ is in one component of the constraint set and the optimum $\mathbf{x}^*$ in another component, then the routine cannot move from $\mathbf{x}^0$ to $\mathbf{x}^*$ while keeping intermediate points feasible. Thus, the most that general MP computer routines guarantee is a local minimum.

One of the most sophisticated routines at NELC is the Ricochet Gradient[26] method. This is an IBM SHARE routine which requires that the

†If the constraint set is bounded, then these conditions are sufficient for a global optimum to problem (1) to exist. This existence theorem gives no method for finding the optimum, which can be difficult in the general problem.

gradients of both the objective function and the constraints exist and be continuous. The method requires an initial feasible point and begins by moving down the gradient of the objective function until a constraint is reached. The program "ricochets" and traverses on the objective function surface across the feasibility region to the opposite constraint. A triangle is then constructed with this traverse line as its base and its apex in the direction of the gradient of f. The next step is made along the line from the base to the apex. The method terminates either on a small step size or when no ricochet is possible.

The user must supply codes to calculate the cost function and the constraints and their gradients. An initial feasible point must also be provided. The program has no options for restarts or postoptimal analysis; in such cases the problem is simply rerun – the known best point is used as initial data.

This routine is capable of producing a tremendous amount of output information. Once the method for controlling this output has been mastered, the user has access to a variety of information, which can be of great use in solving a nonlinear problem. The accompanying user's guide[26] provides detailed documentation on the code and the underlying method. A supplemental user's guide (Appendix 2) reports the results of some test examples and gives a sample deck setup for output control. This program has proved reliable and, after a bit of experience, easy to use.

A second computer code for the general problem is NELC FESDIR. This program is not as sophisticated as the Ricochet Gradient routine, but it can easily be modified for special applications. The user's guide, complete with results on test examples, appears in Appendix 2.

The final code available for solving the nonlinear problem is Experimental SUMT, which was written at Research Analysis Corporation by G. P. McCormick, et al.[27] SUMT is not a production code and is primarily used as a research tool in MP. The experimental nature does not lessen its accuracy, but only its efficiency and speed. The code has modular structure, which allows for easy user modification and adaptation. The user's guide is complete with test examples, although some handwritten corrections and deletions are not too clear. The user has the option of providing an initial point himself or allowing the program to find one. The option also exists of having SUMT compute the gradients by a differencing scheme; the code can also check user-supplied gradients for errors. SUMT provides timing information and allows for user-controlled output. However, the output can be confusing, with the values of different variables appearing under the same headings. The only error message other than incorrectly entered data is a warning during computation that certain estimates indicated the problem functions to be not convex. The theoretical background for SUMT is described in Fiacco and McCormick.[24]

In DESIGNING WITH MATHEMATICAL PROGRAMMING AS AN AID, SUMT and Constraint Transformation, we discuss methods for transforming

a constrained problem into a sequence of unconstrained problems (also called SUMT). This requires more work on the part of the user but can also give him more control in solving the problem and perhaps more insight as to what is happening. In solving the associated unconstrained problems, the user has the option of selecting a direct search method, thereby eliminating the need for differentiable or, in some problems, continuous functions. Special-purpose MP routines can be closely tailored to fit special applications via these techniques.

INTEGER PROGRAMMING

In addition to the standard linear programming problem, there are several special programs under the heading of integer linear programming (ILP). The problem statement is similar except for constraints placed on the variables:

$$\text{Minimize } f(\mathbf{x}) = c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

subject to

$$g_1(\mathbf{x}) = a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leqslant b_1$$

$$g_m(\mathbf{x}) = a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leqslant b_m \qquad (8)$$

$$x_i \geqslant 0 \text{ and } x_i \text{ is an integer.}$$

There are further distinctions within ILP – pure integer, mixed integer, and $\{0,1\}$. The above problem is a pure integer problem; if $x_1, \ldots, x_k$ are required to be integral and $x_{k+1}, \ldots, x_n$ are not necessarily integral, then we have a mixed integer problem; and finally if x_i can equal only zero or one, we have a $\{0,1\}$ ILP. Integer programming is in its infancy, and some methods, although they theoretically exhibit finite convergence, have not been computationally successful. The three ILP routines available at NELC are "Zero-One Integer Programming with Heuristics,"[28] BBMIP,[29] and OPTALG. Zero-one and BBMIP are SHARE routines which have been checked out on the 360/65 but not tested extensively. BBMIP (a mixed integer routine) has been tried on a series of test problems[30] and compared with other routines; however, the other codes are machine-dependent, so the comparison is not meaningful.

OPTALG is a bound and scan pure integer programming code which has solved large problems successfully. It was developed at Stanford by F. S. Hillier,[31] and is currently operational at NELC. The routine requires 336k of core and a solution to the associated linear programming problem; that is, we simply drop the restriction that x_i be an integer. This solution, together

with the LP basis matrix and a guess at an initial feasible solution are then used as data for OPTALG. If the problem is large (a maximum of 61 rows and 61 columns), then this data preparation can be tedious if done by hand. The procedure has been somewhat automated; the exact details are in the user's guide (Appendix 2).

The user should be cautious when attempting to solve an ILP with an integer programming code, since it is possible for a routine to solve some problems and not others, even if they are the same dimension and fairly similar. Matching the routine to the problem is still an art. Progress is being made in this area, but it will be some time before methods are available to solve a general ILP. A selection of engineering applications is presented in Appendix 1.

The following graph (fig. 4)[32] gives an idea of the sizes of mathematical programming problems that can currently be solved. The abscissa represents the sum of both the number of constraints and the number of variables; integer programming methods are still too inconsistent to include.

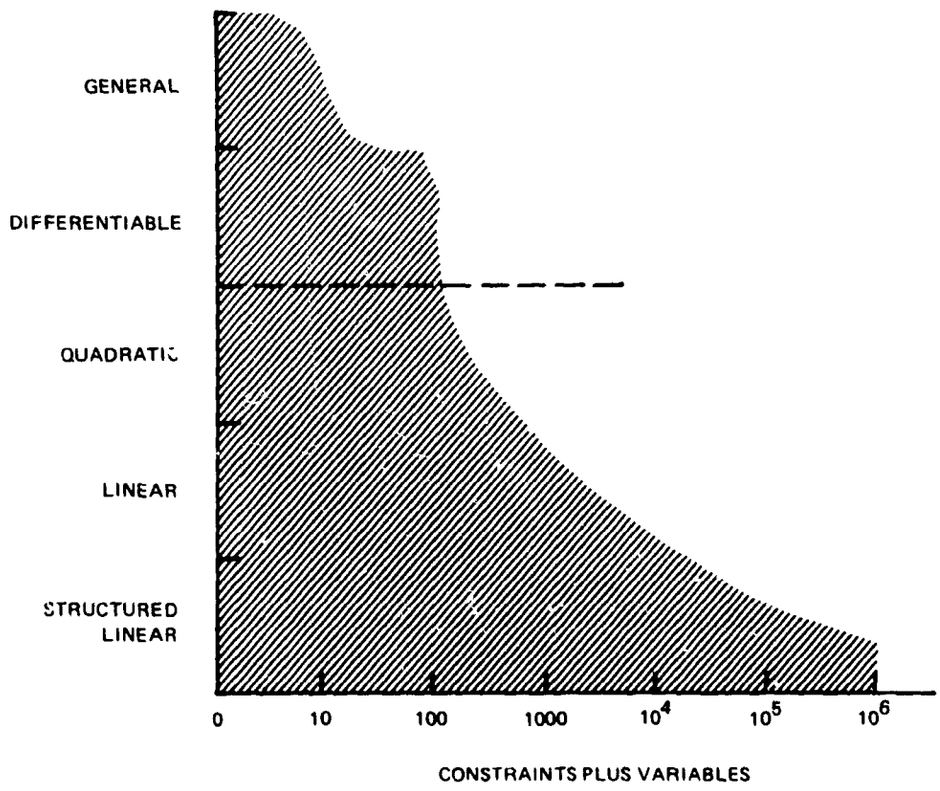


Figure 4. Magnitudes of currently solvable MP problems.

DESIGNING WITH MATHEMATICAL PROGRAMMING AS AN AID

The engineer can effect a better design in some areas in less time and at lower cost with mathematical programming techniques than with classical methods. The key to success is in the word "aid," for, in using the methods effectively, the engineer must have command of his own field and understand some basic principles of MP techniques. It is recognized that typically the user of MP techniques is concerned with obtaining results quickly without lengthy excursions into numerical analysis or mathematical programming. Commercial routines – ECAP,[33] MATCH,[34] etc. – are geared to such a user; however, blindly accepting the output of such routines can be disastrous.[35] Also, these off-the-shelf routines are procrustean – they generally do not lend themselves to modification for a special case. General mathematical programming computer codes can be modified for particular requirements. For example, if filter design by MP techniques is a frequent task, then the computer code can be modified to incorporate the automatic scaling of variables.

In these remaining sections, we discuss how an engineer would proceed from start to finish in using MP as a design aid. We also discuss some special topics, such as duality, that do not apply in all instances, but, if used properly, can save time and money, both in setup analysis and in obtaining a numerical solution.

FORMULATING A MATHEMATICAL PROGRAMMING PROBLEM

In many design problem statements there is, instead of a single goal, a collection of specifications to be satisfied. This gives the engineer several degrees of freedom in formulating an associated MP problem. He has the option of merely satisfying all the design specifications (this is equivalent to finding an initial feasible point) or of singling out one distinguished requirement and using it as the objective function. For example, in reference 1, a tunable bandpass filter was to be designed which would satisfy the following (among other) specifications:

1. At the tuned frequency F, the "insertion loss" of the filter was to be less than 2 dB.

2. At 10% on either side of F, the "roll off" was to be at least 40 dB.

Either of the above options for formulating an MP problem would have been satisfactory. The latter method was chosen. The insertion loss of the filter was selected to be minimized and the roll off requirements were treated as constraints.

To properly formulate an MP problem, the user must explicitly determine the following:

1. The objective (performance, tolerance, etc.) that is to be accomplished.

2. The mathematical relations that govern the interaction between the independent design variables.

3. The bounds and limitations on the values of the components that guarantee a realizable design.

With this information in hand the designer can select which MP approach to use. However, since care is required in choosing the objective function and providing a code for its numerical evaluation, we present some general examples and guidelines.

The objective function can be defined as a measure of the merit or the desirability of a solution to a problem, and its magnitude typically represents cost, profit, performance, quality, etc., or a combination of these. The case of the single well defined objective function generally poses no problems; it is the combination of goals which can lead to difficulties. The following examples illustrate various treatments of multiple goals.

Suppose that it is desired to minimize both the insertion loss of a network denoted by $f_1(x)$ and the cost of the components denoted by $f_2(x)$. Then one formulation of an objective function f would be

$$\text{Minimize } f(x) = f_1(x) + r*f_2(x) \tag{9}$$

where r is an appropriately chosen scaling factor.

Another possible choice of f would be

$$\text{Minimize } f(x) = -f_2(x)/f_1(x) \tag{10}$$

Note that no scaling factor is required and the dimensions of the objective function are

$$-\text{cost(\$)/(unit power loss)} = \text{cost(\$)/(unit power gain)} \tag{11}$$

The next example illustrates treating secondary design goals as constraints. Suppose that high reliability ($f_3(x)$) is desired but the primary goal is a design of minimum costs($f_2(x)$). A constrained formulation would be

$$\text{Minimize } f_2(x) \tag{12}$$

subject to

$$-f_3(x) + \alpha \leq 0$$

where α is a tolerance on reliability (mean time to failure).

A possible numerical pitfall is combining design goals in a haphazard manner. Suppose we have two performance indicators $u_1(x)$ and $u_2(x)$, and we desire to maximize u_1 and minimize u_2 in the same design. Since maximizing u_1 is equivalent to minimizing $-u_1$, a possible objective function

would be

$$\text{Minimize } f(x) = w_2 * u_2(x) - w_1 * u_1(x) \tag{13}$$

where the constants w_1 and w_2 are required to make the function dimensionally correct. These weights are important, since the magnitudes of $-u_1$ and u_2 at the optimum x^* can be quite different. For example, if $-u_1(x^*) \approx 0$ and $u_2(x^*) \approx 1000$, then the effect of u_1 is obliterated by u_2.

The above discussion of objective functions is intended to be general. Aoki[36] presents many detailed examples of engineering applications together with ample background material.

Some care should be taken in the numerical evaluation of the problem functions and their gradients. Coding the functions offers an opportunity for considerable analysis and clever programming. This task is done only once, against the many times that the functions are evaluated during the optimization. A seemingly innocent equation or a naive way of combining terms can lead to poor numerical results. The objective function can be complicated, and a single evaluation for a set of parameters can involve:

1. A solution of a system of differential equations
2. Inverting a matrix
3. Table lookups or interpolation
4. All of the above

Gear[37] and Calahan[38] discuss similar numerical problems and some useful techniques in applying MP to engineering design. One of the goals of mathematics as applied to computer-aided design is to free the applications-oriented user from the standard numerical worries. For example, the routine should be able to analyze the problem and choose the best method for integrating a system of nonlinear differential equations. State-of-the-art techniques do not meet this goal; thus, it is still up to the user to make the proper selection of numerical methods.

Duality is a case in which careful analysis can pay off generously. Full details with examples are presented in the section on duality, but we present this idea here to point out some analysis and numerical considerations. In some applications of linear programming, problems occur which have a much larger number of constraints than variables. To be specific, we may have 50 constraints and 10 variables. To obtain a solution to the LP problem posed in this way, an LP code would essentially invert a 50-by-50 matrix. This inversion can be time-consuming and perhaps inaccurate for such a large matrix. We show in Duality in MP how to cast this problem as a dual linear programming problem which would require only a 10-by-10 matrix to be inverted. Duality is an excellent example of a little analysis saving a great deal of time and effort.

Suppose that the design problem is clearly stated as an MP problem. In order to solve the resulting problem, it may be necessary to transform it into an equivalent MP problem. The possible modifications can occur in the light of the following questions:

1. Is the necessary computer code available?
2. Would duality aid in obtaining numerical results?
3. Is an initial point available?
4. Can the derivatives be easily calculated?
5. Would a postoptimal analysis be useful?
6. Can significant benefit be gained by scaling the variables or clever coding techniques?

It does little good to cast a design problem as an MP problem, if we lack the numerical means to solve it. Thus, the user should be prepared to transform his problem into a solvable form or into a more useful form. In the ensuing pages we address ourselves to problem modifications which we found useful in practice.

SUMT AND CONSTRAINT TRANSFORMATION

The most common mismatch is that of a constrained problem to be solved and a routine for solving only unconstrained problems. The sequential unconstrained minimization techniques (SUMT) developed by Fiacco and McCormick[24] transform a constrained MP problem into an equivalent sequence (in terms of having the same solution) of unconstrained problems. The transformation takes place with the aid of an unconstrained auxiliary function which has the following form:

$$\phi(x, r_k) \equiv f(x) + p(r_k) \sum_{i=1}^{m} G(g_i(x)) \tag{14}$$

where r_k is a parameter, G(y) is a monotonic function of y that behaves in some well chosen manner at y = 0, and p(r) is a function of r which depends on the choice of G. Typical choices require $G(y) > 0$ for $y > 0$ and $G(y) = 0$ for $y < 0$, or require that G(y) approach ∞ as y approaches 0 through values less than zero. The first choice of G is usually associated with procedures that are not concerned with constraint satisfaction except at the solution (exterior methods); and the second choice of G is associated with procedures which enforce constraint satisfaction throughout the minimization (interior methods). The basic idea of SUMT is the following. Let x^* be a solution to problem 1; that is, we assume x^* minimizes $f(x)$, subject to $g_i(x) \leq 0$ for i=1, . . ., m. Then under appropriate conditions[24] on the problem functions the following theorem holds.

If $\{x_k\}_{k=1}^{\infty}$ is a sequence of points, each of which minimizes $\phi(x, r_k)$ where r_k is a sequence of points tending to zero, then we have the limit $x_k = x^*$, or in some cases limit $f(x_k) = f(x^*)$. So a constrained problem is replaced by an equivalent sequence (in the sense of having a common solution) of unconstrained problems. Common choices for G(y) are $-1/y$, $(\min [y, 0])^{1+\epsilon}$, $\epsilon > 0$, and $-\log(-y)$ with $p(r) = r$, $1/r$, r respectively. For the first form of G(y), $\phi(x, r_k)$ becomes

$$\phi(x, r_k) = f(x) + r_k \sum_{i=1}^{m} \left\{-1/g_i(x)\right\} \tag{15}$$

then if we have a feasible point x° and seek to minimize $\phi(x, r_k)$, the term $\Sigma\ 1/-g_i(x)$ keeps intermediate test points in the constraint set. For, if the routine attempts to leave the feasible set, it must cross the boundary; this causes $\Sigma\ 1/-g_i(x)$ to approach $+\infty$, and, since we are minimizing, the routine automatically avoids points which yield large values of $\phi(x, r_k)$. As r_k approaches zero, x_k is approaching x^*, and since x_k is used as the initial point to find x_{k+1}, each successive minimization requires fewer iterations. In Lagrange Multipliers as a Design Aid we give some examples.

Variations of the above techniques can be applied in many situations where there is a mismatch between computer code and mathematical programming problem. For example, Rosen's[22] gradient projection method solves the following problem:

$$\text{Minimize } f(x) \tag{16}$$

Subject to

$$\sum_{j=1}^{n} a_{ij} x_j \leqslant b_i \qquad i = 1, 2, \ldots, m$$

Now suppose the MP problem we have on hand has some nonlinear constraints $g_i(x) \leqslant 0$, $i = 1, \ldots, \ell$, as well as some linear ones. We modify the problem as follows:

$$\text{Minimize } f(x) - r_k \sum_{i=1}^{\ell} 1/g_i(x) \tag{17}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leqslant b_i \qquad i = 1, \ldots, m$$

Thus, we take advantage of a good routine which our problem does not quite fit.

Another method which allows us to make use of an unconstrained computer code is a straight constant transformation. Many times in Problem 1 the constraints are fairly simple – for example, $a_i \leqslant x_i \leqslant b_i$ or $c_i \leqslant a_i x_i + b_i \leqslant d_i$ – and the objective function is very complicated with a difficult gradient to compute. The simple nature of the above constraints allows an initial feasible point to be obtained immediately, and the constraint transformation keeps the intermediate points feasible. This transformation (as well as SUMT) allows a direct search method to be used for optimizing complicated objective functions. However, as the examples show, there is a point of diminishing return in trying to transform all the constraints, since, as the constraints become complicated, it can be impossible to find a well mannered constraint transformation to do the job.

If we require $a \leqslant x \leqslant b$, then the following transformations keep x within this range:

1. $x = \frac{b+a}{2} + \frac{(b-a)}{2} \sin y$
2. $x = a + (b-a) \sin^2 y$
3. $x = a + (b-a) \frac{e^y}{e^y + e^{-y}}$ for y unconstrained.

For one-sided boundaries – that is, $a \leqslant x$ – we have:

4. $x = a + e^y$
5. $x = a + |y|$
6. $x = a + y^2$

The next types of constraints are linear inequalities – for example, $d < ax + b < c$. However, this is equivalent to a boundary inequality; that is, $\frac{d-b}{a} < x < \frac{c-b}{a}$ for $a > 0$.

If we have two linear inequalities in two unknowns, a transformation is still possible:

$$a \leqslant b_{11}x_1 + b_{12}x_2 \leqslant c$$

$$e \leqslant b_{21}x_1 + b_{22}x_2 \leqslant f$$

then let $y_1 = b_{11}x_1 + b_{12}x_2$

$$y_2 = b_{21}x_1 + b_{22}x_2$$

then $x_1 = (b_{22}y_1 - b_{12}y_2)/D$

$$x_2 = (b_{21}y_1 - b_{11}y_2)/D$$

where $D = b_{22}b_{11} - b_{12}b_{21}$

then let $y_1 = a + (c-a) \sin^2 z_1$

$y_2 = e + (f-e) \sin^2 z_2$

Thus, we have made x_1, x_2 functions of the unrestricted variables z_1, z_2, and they still satisfy the inequality constraints.

OTHER EXAMPLES

1. Suppose we require $0 \leq x_1 \leq x_2 \leq x_3$

then consider $x_1 = y_1^2$

$x_2 = y_1^2 + y_2^2$

$x_3 = y_1^2 + y_2^2 + y_3^2$

2. This example was presented in Box[39]

maximize $f = [9 - (x_1 - 3)^2] \, x_2^3/27\sqrt{3}$

subject to $0 \leq x_1$

$0 \leq x_2 \leq x_1/\sqrt{3}$

$0 \leq x_1 + \sqrt{3}\,(x_2) \leq 6$

Initial point

$x_1 = 1, x_2 = .5$ $\quad$ $f = .01336$

optimum $x_1 = 3, x_2 = \sqrt{3}$ $\quad$ $f = 1.0$

TRANSFORMATION OF CONSTRAINTS

$y_1 = x_1 + \sqrt{3}\, x_2$

$y_2 = x_2 - x_1/\sqrt{3}$

Then if $x_2 = x_1/\sqrt{3}$, we have $0 \leq x_1 + x_1 \leq 6$ or $x_1 \leq 3$. This then gives bounds for the variables y_1 and y_2.

$-x_1/\sqrt{3} \leq x_2 - x_1/\sqrt{3} \leq 0$

$-\sqrt{3} = -3/\sqrt{3} \leq y_2 \leq 0$. Thus we have

$$0 \leq y_1 \leq 6$$

$$-\sqrt{3} \leq y_2 \leq 0.$$

This implies

$$x_1 = 1/2\,(y_1 - \sqrt{3}y_2)$$

$$x_2 = 1/2\,(y_1/\sqrt{3} + y_2)$$

Thus, we have the unconstrained problem

maximize $f(x_1(z_1, z_2), x_2(z_1, z_2))$

where $x_1 = 1/2\,(y_1 - \sqrt{3}y_2)$

$x_2 = 1/2\,(y_1/\sqrt{3} + y_2)$

$y_1 = 6 \sin^2 z_1$

$y_2 = -\sqrt{3} + \sqrt{3} \sin^2 z_2$

The necessary FORTRAN code to evaluate this objective function, if we were using NELC DIRECT to solve the minimization problem, is as follows:

```
      SUBROUTINE FN(Z, N, F)
      DIMENSION Z(2)
      Y1 = 6.0*(SIN(Z(1)))**2
      Y2 = -2.0*SQRT(3)*(1-(SIN(Z(2)))**2)
      X1 = .5*(Y1-SQRT(3)*Y2
      X2 = .5*(Y1/SQRT(3)+Y2)
      F = (9-(X1-3.)**2)*(X2**2)/(27.0*SQRT(3))
      RETURN
      END
```

The best transformation to use depends both on the problem and the minimization scheme used. For example, transformation 5 is not differentiable at $y=0$; hence, a gradient method would not be valid at this point. Method 3 would make coding the gradient of the objective function lengthy and time-consuming to debug. These problems do not occur if a direct search method is used.

GRADIENT APPROXIMATION

Let us now suppose the problem is matched to the routine and further suppose the computer code is a gradient method. The user must supply a subroutine or function that will compute the gradient; that is, he must provide the following vector:

$$\nabla f^T(x) = \left(\frac{df}{dx_1}, \frac{df}{dx_2}, \ldots, \frac{df}{dx_n}\right) \tag{18}$$

Often, the complexity of the objective function f(x) or of some of the constraint function is such that the gradient cannot be computed, or the time required to derive ∇f analytically is excessive. Then, the designer, in order to use the gradient methods, can decide to approximate the gradient. In the process, the function must be evaluated several times in the vicinity of a point x for each approximation. Thus, a trade-off situation arises, since the more accurate estimate will require many function evaluations which may ultimately be more costly (in computer time) than direct calculation of the gradient (in manhours). The problem becomes one of selecting the most accurate approximation for the least number of function evaluations.

We list some of the standard schemes for approximating the derivative of a function together with the corresponding error estimates (Hildebrand[40]). Let

$$f_1 = f(x_1, x_2, \ldots, x_i + \Delta, \ldots, x_n) \tag{19}$$

$$f_0 = f(x_1, x_2, \ldots, x_i, \ldots, x_n) \tag{20}$$

$$f_{-1} = f(x_1, \ldots, x_i - \Delta, \ldots, x_n) \tag{21}$$

$$\epsilon = \frac{\partial^3 f}{\partial x_i^3}(x_1, \ldots, x_i + \xi, \ldots, x_n) \text{ where } |\xi| < \Delta \tag{22}$$

Then $$\partial f_1/\partial x_i = (1/2\Delta)(-3f_{-1} + 4f_0 - f_1) + \frac{\epsilon\Delta^2}{3} \tag{23}$$

$$\partial f_0/\partial x_i = (1/2\Delta)(f_{-1} + f_1) - \frac{\epsilon\Delta^2}{6} \tag{24}$$

$$\partial f_{-1}/\partial x_i = (1/2\Delta)(f_{-1} - 4f_0 + 3f_1) + \frac{\epsilon\Delta^2}{3} \tag{25}$$

We note that the best estimate for $\partial f/\partial x_i$ is equation (29), for if $|\partial^3 f/\partial x_i^3| < M_3$ as ξ varies over $|\xi| < \Delta$ we have the maximum error $|\epsilon \max| = M_3/6$. It is interesting that the point in question does not appear in the formula, even though f_0 is generally available. This additional information does not improve the estimate. The next question is, how should Δ be chosen?

Hildebrand gives an optimal Δ which is derived analytically via $\partial^3 f(x)/\partial x_i^3$. Unfortunately, this expression is generally not available when we are attempting to approximate $\partial f(x)/\partial x_i$. Also, this optimal Δ is a function of the test point x, which changes many times in the course of a minimization. If this approach is taken, then an educated guess will have to be made for Δ. Experiments have indicated that for the most efficient operation, Δ should be changed automatically by the code. A method along this line is Stewart's[41] modification of Davidon's[42] minimization routine.

Steward's method is dependent on the information generated by the minimization part of the routine. Using this information, he is able to select a good Δ_i and initially estimate the ith component of the gradient by the formula $(f_1 - f_0)/\Delta_i$. When this simple scheme begins to fail, the routine automatically switches to the central difference method for a more accurate estimate. Δ_i is also suitably modified. His method required 163 function evaluations to find the minimum of the Rosenbrock function to within five decimal places. This compares with 325 for the direct search method (ZANGWL) and 71 for the gradient method (CNJGAT), to minimize to approximately the same accuracy. Unfortunately, Steward's method was coded in a language for the CDC 1604 and has not been modified for the NELC IBM 360.

The final technique for general numerical differentiation which we discuss, is a code from the IBM Scientific Subroutine Package[11], called DDCAR. DDCAR uses an extrapolation technique to obtain highly accurate estimates of the gradient. This method does not need an optimal Δ to give good results. However, we must pay the price of using a larger number of function evaluations than the central difference method would require, for each estimate of ∇f. A code and results of using DDCAR to compute the gradient in minimizing the Rosenbrock function with the gradient method NELC CNJGAT are presented in table 1. 1850 function evaluations were required to obtain these results.

The methods presented above are general and can be used for a wide class of functions. Some applications may lend themselves to special methods for estimating the gradient. For example, Calahan[38] discusses methods for numerically evaluating the gradient in a network optimization scheme. The method is highly specialized for this type of problem. He uses a "calculus of variations" approach together with numerical integration. The section on gradient calculation is a good example of the success of careful analysis.

```
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON KOUNT
      DIMENSION X(30), H(30,30), G(30), S(30), SIGMA(30),XX(20)
      EXTERNAL FG BOX
      X(1)=-1.2D00
      X(2)=-1.0D00
      KOUNT= 0
      NMAX = 30
      MPRNT = 1
      N=  2
      ISTART = N + 1
      EPSLON = 1.0D-10
      FEST = 0.0
      ITERBD = 80
      SBOUND = 1.0D-09
      ICNT = 0
      IREST = 0
      ITER = 0
      NM1 = N - 1
      DO 30 I = 1, NM1
      H(I,I) = 1.0
      IP1 = I + 1
      DO 30 J = IP1, N
      H(J,I) = 0.0
30    H(I,J) = 0.0
      H(N,N) = 1.0
      DO 20 J=1,20
20    S(J) = 0.
      CALL CNJGAT(FGBOX,N,NMAX,X     ,ITERBD,EPSLON,FEST,MPRNT,
     ISTART,ITE 1 R,F,G,S)
      PRINT 999,KOUNT
999   FORMAT( 1X,'KOUNT',I5)
      END

      FUNCTION USERF (X,N)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(N)
      T1=X(2)-X(1)*X(1)*X(1)
      T2=X(1)-1.0D00
      USERF=100.0D0C*T1*T1 + T2*T2
      RETURN
      END

      FUNCTION  FUNC(H)
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON KOUNT/GRAD/XX,IVAR,NN
      DIMENSION XTEMP(20) ,XX(20)
      DO 30 J = 1,NN
      IF (J .EQ. IVAR) GO TO 29
      XTEMP(J)  = XX(J)
```

```
      GO TO 30
29    XTEMP (IVAR)  =  XX(IVAR) + H
30    CONTINUE
      FUNC=USERF(XTEMP,NN)
      KOUNT = KOUNT + 1
      RETURN
      END

      SUBROUTINE FG BOX( X ,F , G, N)
      IMPLICIT REAL*8 (A-H,O-Z)
      EXTERNAL FUNC
      DIMENSION X(1),G(1) ,XX(20)
      COMMON/GRAD/XX,IVAR,NN
      NN      =  N
      DO 10 J=1,N
10    XX(J)   = X(J)
      IVAR    = 1
      F       =  FUNC(0)
      DO 20 K=1, N
      IVAR    = K
      WZ =  .001
      CALL DDCAR(0.,WZ ,1,FUNC,Y)
20    G(K)    = Y
      RETURN
      END
```

ITERATION	F	
	0.578384000D 02	
1	0.405290420 01	X(1) =-0.101177240 01 X(2) =-0.10432685D 01
2	0.405138330 01	X(1) =-0.10125637D 01 X(2) =-0.10412820D 01
3	0.388294000 01	X(1) =-0.932513850 00 X(2) =-0.772333670 00
4	0.368093350 01	X(1) =-0.915208780 00 X(2) =-0.777947070 00
5	0.368073090 01	X(1) =-0.918470470 00 X(2) =-0.773389670 00
6	0.367882320 01	X(1) =-0.917157350 00 X(2) =-0.765720770 00
7	0.366215030 01	X(1) =-0.912331470 00 X(2) =-0.766546360 00
8	0.333074130 01	X(1) =-0.782367800 00 X(2) =-0.439655940 00
9	0.331770710 01	X(1) =-0.733347000 00 X(2) =-0.338426770 00
10	0.290635270 01	X(1) =-0.699968520 00 X(2) =-0.355783280 00
11	0.289191870 01	X(1) =-0.700544840 00 X(2) =-0.344613740 00
12	0.238089900 01	X(1) =-0.519995180 00 X(2) =-0.114055230 00
13	0.225754770 01	X(1) =-0.488185800 00 X(2) =-0.137047480 00
14	0.215807320 01	X(1) =-0.468332700 00 X(2) =-0.981709060-01
15	0.185778300 00	X(1) = 0.642544540 00 X(2) = 0.241199180 00
16	0.135665720 00	X(1) = 0.632073720 00 X(2) = 0.250803940 00
17	0.989754120-01	X(1) = 0.704499570 00 X(2) = 0.338861190 00

18	0.70097588D-01	X(1) = 0.799033230 00 X(2) = 0.492909470 00
19	0.43186873D-01	X(1) = 0.792359390 00 X(2) = 0.496619690 00
20	0.37085901D-01	X(1) = 0.871943160 00 X(2) = 0.648542090 00
21	0.37085901D-01	X(1) = 0.871943160 00 X(2) = 0.648542090 00
22	0.17754933D-01	X(1) = 0.866842970 00 X(2) = 0.650869010 00
23	0.10469106D-01	X(1) = 0.920942610 00 X(2) = 0.774588510 00
24	0.10043283D-01	X(1) = 0.933755520 00 X(2) = 0.806620900 00
25	0.47209678D-02	X(1) = 0.931328900 00 X(2) = 0.807580950 00
26	0.19939622D-02	X(1) = 0.963284400 00 X(2) = 0.891306290 00
27	0.18329659D-03	X(1) = 0.995003160 00 X(2) = 0.983825970 00
28	0.28885366D-04	X(1) = 0.994627940 00 X(2) = 0.983954010 00
29	0.50345410D-06	X(1) = 0.999818620 00 X(2) = 0.999386990 00
30	0.12440066D-07	X(1) = 0.999889020 00 X(2) = 0.999668220 00
31	0.12228058D-07	X(1) = 0.999889470 00 X(2) = 0.999668110 00
32	0.13578763D-12	X(1) = 0.100000000 01 X(2) = 0.999999990 00
33	0.87952086D-18	X(1) = 0.100000000 01 X(2) = 0.100000000 01
34	0.87854680D-18	X(1) = 0.100000000 01 X(2) = 0.100000000 01
35	0.24394397D-23	X(1) = 0.100000000 01 X(2) = 0.100000000 01
36	0.59070991D-27	X(1) = 0.100000000 01 X(2) = 0.100000000 01

DUALITY IN MP

In linear programming and certain SUMT methods with convex functions, the possibility of using duality to gain computational efficiency should be considered.

Duality occurs in many areas – mathematics, engineering, economics physics, etc. In a mathematical or engineering context it implies that two concepts or systems have a specific mathematical relationship. We give an example from circuit theory before proceeding with duality and MP.

Consider the following series RCL network (fig. 5).[16]

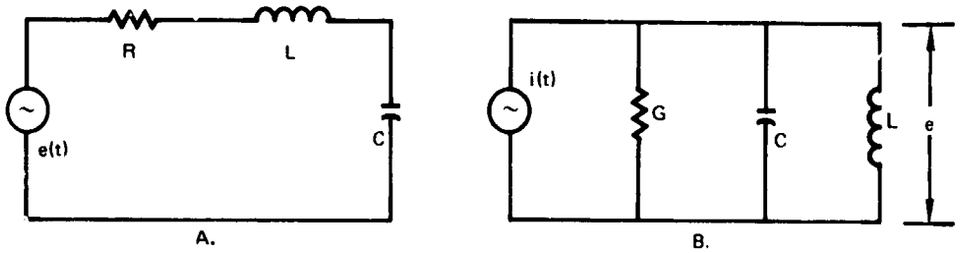


Figure 5. Dual circuits.

The mathematical equation representing (A) is

$$L\frac{di}{dt} + Ri + \frac{1}{C}\int_0^t i\,d\tau = e(t) \tag{26}$$

If we interchange e and i and L and C and replace R by G = 1/R, we have

$$C\frac{de}{dt} + Ge + \frac{1}{L}\int_0^t e\,d\tau = i(t) \tag{27}$$

which is the equation which models the parallel network (B). Thus, an equation of the form

$$a\frac{dx}{dt} + bx + c\int_0^t x\,d\tau = y(t)$$

can represent either circuit (A) or circuit (B) depending on the parameters a, b, c; that is, it has a dual function, and we say that (A) is the dual of (B).

In linear programming, we have the most straightforward application and complete theory of duality. Let x and c be vectors of length n, b a vector of length m, and A an m-by-n matrix, Then the linear programming problem seeks to find the vector x which produces

a minimum $z = c^T x$

subject to

$$Ax \leqslant b \qquad (28)$$

$$x > 0.$$

To this linear programming problem there corresponds an associated problem.

Maximize $v = b^T w$

subject to

$$A^T w \geqslant c \qquad (29)$$

$$w \geqslant 0$$

where w is an m vector. Problem (28) is called the primal problem while problem (29) is referred to as the dual problem.

In the primal formulation we have m constraints and n variables, and just the opposite in the dual formulation. Some facts about these corresponding problems which are pertinent to this section are:

1. The primal problem has a bounded solution x^* if and only if the dual has a bounded solution w^*.
2. w^* can be obtained from x^* and vice versa.
3. The dual of the dual is the primal.
4. In the case of bounded solutions we have $z^* = v^*$.

The efficiency of linear programming computer codes decreases as the number of constraints increases. It is this trait which makes the study of the duality relationship worthwhile from a computational standpoint. The following example shows how this loss of efficiency can be lessened by judicious use of the dual.

Consider the following programming problem:

Minimize $z = -4x_1 - 3x_2$

subject to

$$\begin{aligned} x_1 &\leqslant 6 \\ x_2 &\leqslant 8 \\ x_1 + x_2 &\leqslant 7 \\ 3x_1 + x_2 &\leqslant 15 \end{aligned} \qquad (30)$$

$$-x_2 \leq 1$$

$$x_1 \geq 0,$$

Note that x_2 can be negative. To pose this as a standard †LP problem, we replace the variable x_2 by the difference of two positive variables; that is, we let $x_2 = x_2' - x_2''$. Now the problem can be written in standard form.

$$\text{Minimize } -4x_1 - 3(x_2' - x_2'')$$

subject to

$$\begin{aligned} x_1 &\leq 6 \\ x_2' - x_2'' &\leq 8 \\ x_1 + x_2' - x_2'' &\leq 7 \\ 3x_1 + x_2' - x_2'' &\leq 15 \\ -x_2' + x_2'' &\leq 1 \\ x_1, x_2', x_2'' &\geq 0 \end{aligned} \tag{31}$$

Writing the constraints in matrix notation, we have

$$Ax = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 1 & 1 & -1 \\ 3 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2' \\ x_2'' \end{bmatrix} \leq \begin{bmatrix} 6 \\ 8 \\ 7 \\ 15 \\ 1 \end{bmatrix}$$

In solving this LP problem, we add another positive variable xs_i, called a slack variable, to each row, to make each inequality an equality. The problem then becomes:

†MPS/360 accepts unrestricted variables and automatically makes this transformation.

$$\text{Minimize } -4x_1 - 3x_2' + 3x_2'' + 0xs_1 + \ldots + 0xs_5$$

subject to

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 1 & 0 & 0 \\ 3 & 1 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2' \\ x_2'' \\ xs_1 \\ xs_2 \\ xs_3 \\ xs_4 \\ xs_5 \end{bmatrix} = \begin{bmatrix} 6 \\ 8 \\ 7 \\ 15 \\ 1 \end{bmatrix} \tag{32}$$

Note that each slack variable appears in the cost row with zero for a coefficient.

The simplex method of LP performs an iterative process on five selected columns of this augmented matrix to obtain a basis inverse matrix. This matrix is then used to obtain the solution. It is this inversion process which causes the efficiency of LP codes to decrease as m increases. We now cast problem (31) in its dual formulation, which will reduce the number of rows.

$$\text{Maximize } v = 6w_1 + 8w_2 + 7w_3 + 15w_4 + w_5$$

subject to

$$A^T w = \begin{bmatrix} 1 & 0 & 1 & 3 & 0 \\ 0 & 1 & 1 & 1 & -1 \\ 0 & -1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} \geqslant \begin{bmatrix} -4 \\ -3 \\ 3 \end{bmatrix} \tag{33}$$

Now from each row we subtract a positive variable ws_i and write the constraint matrix as:

$$\begin{bmatrix} 1 & 0 & 1 & 3 & 0 & -1 & 0 & 0 \\ 0 & 1 & 1 & 1 & -1 & 0 & -1 & 0 \\ 0 & -1 & -1 & -1 & 1 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ ws_1 \\ ws_2 \\ ws_3 \end{bmatrix} = \begin{bmatrix} -4 \\ -3 \\ 3 \end{bmatrix}$$

We now have a formulation similar to problem (33); however, to solve this problem, it is necessary to "invert" only a 3-by-3 matrix. Even though the number of variables is the same, the smaller number of constraints makes this formulation more efficient. If we were to solve the dual formulation using MPS/360, since the dual of the dual is the primal, the solution to problem (31) would appear under the DUAL ACTIVITY heading.

There is another form of the primal-dual relationship which has both formulational and computational advantages; it is the unsymmetric form.

The primal problem can also be stated as: find a column vector x^* which

minimizes $z = c^T x$

subject to

$$Ax = b \tag{34}$$

$$x \geqslant 0$$

The original LP problem (28) can be put in this form by adding slack variables to transform the inequality constraints into equality constraints. Then the unsymmetric dual to (34) is

Maximize $v = b^T w$

subject to (35)

$$A^T w \leqslant c$$

We notice in (35) that there is no restriction on the sign of w. This is most useful in using linear programming as an analysis tool, and to condense the problem size. The three previous properties for the symmetric form of the dual also hold for the unsymmetric form.

We return to problem (30) to give a simple example of the usefulness of this unsymmetric form. The problem is written as

Maximize $v = 4w_1 + 3w_2$

subject to

$$A^T w = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 3 & 1 \\ 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \leqslant \begin{bmatrix} 6 \\ 8 \\ 7 \\ 15 \\ 1 \\ 0 \end{bmatrix} \tag{36}$$

Note that w_2 is unrestricted in sign and that the last row of A^T keeps w_1 nonnegative. This is the unsymmetric dual formulation of problem (30). Since w_2 is unrestricted in sign, to solve this problem via a standard LP code, we should have to replace w_2 by $w_2' - w_2''$, as before. But if we consider the associated primal problem, this requirement disappears; i.e.,

$$\text{Minimize } 6x_1 + 8x_2 + 7x_3 + 15x_4 + x_5$$

subject to

$$Ax = \begin{bmatrix} 1 & 0 & 1 & 3 & 0 & -1 \\ 0 & 1 & 1 & 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix} \tag{37}$$

To solve this primal problem we must invert only a 2-by-2 matrix rather than a 5-by-5 as in the dual case. When we solve this program using MPS/360, the optimal w will appear as a DUAL ACTIVITY.

This method accomplishes two things:

1. There is no increase in the number of variables.

2. If m is much larger than n, then the primal has fewer constraints and will generally be faster to solve.

Of course, the form we decide to use will depend on the relative sizes of m and n.

We present another example[43] which will illustrate both the use of the unsymmetric dual and the utility of LP in the area of applied mathematics; viz., linear boundary value problems. Consider the following problem. Find a solution, y, to $L[y] = r(x)$ over $[a, b]$, where

$$L[y] \equiv \sum_{j=0}^{n} f_j(x)y^{(j)} = f_0(x)y + f_1(x)y^{(1)} + \ldots + f_n(x)y^{(n)} \tag{38}$$

with boundary conditions $V_j[y] = \Gamma_j$, $j = 1, 2, \ldots, n$

$$\text{where } V_j[y] \equiv \sum_{k=0}^{n-1} (\alpha_{j,k}y^{(k)}(a) + \beta_{j,k}y^{(k)}(b)). \tag{39}$$

and $y^{(i)}$ is the ith derivative of $y(x)$ with respect to x.

Some boundary value problems do not have a closed form solution or even a solution in the limit; however, we should still like some information and in most instances an approximate solution is sufficient. The approximation is made in the min-max sense; i.e., if f* is the theoretical solution to, say, a differential equation over an interval [a, b], then an approximation f_α is sought to

$$\text{minimize} \quad \left\{ \underset{x \in [a, b]}{\text{maximum}} \; | f_\alpha(x) - f^*(x) | \right\} \tag{40}$$

This formulation lends itself to an application of mathematical programming. We approximate the solution by a sum of functions over the interval [a, b]. To do this, we partition [a, b] as follows:

$$a < x_1 < x_3 \ldots < x_m < b, \; m \gg p \text{ and seek parameters } a_1, \ldots, a_p$$

which minimize

$$\max_{1 \leq i \leq m} \; | L(y^*(x_i)) - r(x_i) | \tag{41}$$

where

$$y^*(x) = y_o(x) + \sum_{j=1}^{p} a_j y_j(x), \tag{42}$$

$y_o(x)$ satisfies $V_r(y_o) = \Gamma_r$, r = 1, . . ., n

and $y_j(x)$ satisfy $V_r(y_j) = 0$. j = 1, . . ., p

Substituting equation (42) into equation (41), and introducing an additional variable ϵ, we wish to find $a_1, \ldots, a_p, \epsilon$ which minimizes ϵ subject to

$$| L (y_o(x_i) + \sum_{j=1}^{p} a_j y_j(x_i)) - r(x_i) | < \epsilon$$

for i = 1, 2, . . . m

For LP to be applied, the constraints must be linear; thus, we have

Minimize ϵ
$(a_1, a_2, \ldots, a_p, \epsilon)$

subject to

$$-\epsilon < \sum_{i=1}^{p} a_i L[y_i(x_j)] + L[y_o(x_j)] - r(x_j) < \epsilon \; j = 1, 2, \ldots m,$$

or, rewriting as two single inequalities,

$$-\sum_{i=1}^{p} a_i L(y_i(x_j)) - L(y_o(x_j)) + r(x_j) \leqslant \epsilon \tag{43}$$

$$\sum_{i=1}^{p} a_i L(y_i(x_j)) + L(y_o(x_j)) - r(x_j) \leqslant \epsilon \tag{44}$$

Collecting all the parameters on the left side of the inequality yields

$$-\sum_{i=1}^{r} a_i L(y_i(x_j)) - \epsilon \geqslant L(y_o(x_j)) - r(x_j)$$

$$\sum_{i=1}^{p} a_i L(y_i(x_j)) - \epsilon \geqslant r(x_j) - L(y_o(x_j))$$

$$\text{for } j = 1, 2, \ldots m$$

where

$a = x_1 < x_2 \ldots < x_m = b$ is a partition of $[a, b]$.

Form the p-by-m matrix:

$$G = \begin{bmatrix} L[y_1(x_1)] & \ldots & L[y_1(x_m)] \\ \vdots & & \vdots \\ L[y_p(x_1)] & \ldots & L[y_p(x_m)] \end{bmatrix}$$

the 2m vector

$$c^T = \left| L(y_o(x_1) - r(x_1), \ldots L(y_o(x_m) - r(x_m), r(x_1) - L(y_o(x_1)), \ldots, \right.$$

$$\left. r(x_m) - L(y_o(x_m)) \right|$$

the (p + 1) vectors

$$\alpha^T = (a_1, \ldots, a_p, \epsilon)$$

$$b^T = (0 \ldots 0, -1)$$

and the matrix

$$A = \begin{bmatrix} G & -G \\ -1, \ldots, -1 & -1, \ldots, -1 \end{bmatrix}$$

Since there is no restriction on the sign of the parameters a_i (i=1, . . ., p), we cast the programming problem in the dual formulation; i.e.,

$$\text{Maximize } -\epsilon = \alpha^T \cdot b \qquad (45)$$

$$(a_1, a_2, \ldots, a_p, \epsilon)^{\dagger}$$

subject to

$$\begin{bmatrix} G^T & \begin{matrix} -1 \\ \vdots \\ -1 \end{matrix} \\ -G^T & \begin{matrix} -1 \\ \vdots \\ -1 \end{matrix} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \\ \epsilon \end{bmatrix} \leqslant \begin{bmatrix} L(y_o(x_1)) - r(x_1) \\ \vdots \\ L(y_o(x_m) - r(x_m) \\ r(x_1) - L(y_o(x_1)) \\ \vdots \\ r(x_m) - L(y_o(x_m)) \end{bmatrix}$$

or more succinctly as

$$\text{Maximize } \alpha^T b$$

subject to (dual)

$$A^T \alpha \leqslant c$$

then, since the dual of the dual is the primal, we have

$$\text{Minimize } c^T x$$

subject to (primal)

$$Ax = b$$

$$x \geqslant 0 \qquad \text{where x is a 2m vector.}$$

This primal problem is then solved numerically and the optimal α vector appears as the DUAL ACTIVITY vector.

---

†Note inequalities (43) and (44) keep $\epsilon \geqslant 0$.

In summary, the basic advantages are as follows: We wish parameters $a_1, \ldots, a_p$, which give the best estimate in a min-max sense of the solution to the linear boundary value problem. The parameters are unrestricted in sign and the number of points m in the interval of solution is large. To cast this problem as a primal linear programming problem would have required $2p + 1$ variables, since the a_i are unrestricted, and $2m$ constraints. By first writing the approximation problem as a dual linear programming problem (45), we need only $p + 1$ variables; then transforming to the primal reduces the number of constraints from $2m$ to $p + 1$. We give two examples which use the above method. Example 1:[43] A homogeneous equation with inhomogeneous boundary conditions

$$L(y) = xy'' - (x+1)y' - 2(x-1)y = 0; y(0) = 1, y(1) = 0 \text{ on } [0, 1]$$

$$y^*(x) = y_0(x) + \sum_{j=i}^{p} a_j y_j(x)$$

with

$$y_0(x) = 1-x; y_j(x) = x^j(1-x)$$

$$j = 1, 2, \ldots, p$$

then

$$L[y_0] = 2x^2 - 3x + 3$$

$$L[y_j] = 2x^{j+2} + (j-3)x^{j+1} - (j^2 + j-3)x^j + j(j-2)x^{j-1}$$

$$x_i = (i-1)(0.05). \qquad i = 1, 2, \ldots, 21$$

for $p = 3$, $a_1 = 3.0374706$

$a_2 = -0.78655970$

$a_3 = 0.88391133$

$\epsilon = 0.03747406$, and the maximum error occurred at $x = 0$.

Example 2:[43] Inhomogeneous case with homogeneous boundary conditions

$$L[y] = y'' + (1+x^2)y = -1, y(1) = y(-1) = 0 \qquad \text{on } [-1, 1]$$

$$y_j(x) = 1 - x^{2j}$$

$$L[y_j] = -x^{2(j+1)} - x^{2j} - 2j(2j-1)x^{2j-1} + x^2 + i$$

Let

$$z = x^2,\ L[y_j] = -z^{j+1} - z^j - 2j(2_j - 1)z^{j-1} + z + 1$$

$$z_i = (i-1)(0.05) \qquad i = 1, 2, \ldots, 21$$

for $p = 3$, $a_1 = 0.9675$, $a_2 = \ldots$ $a_3 = -0.0285$

$\epsilon = 0.0029$

The preceding paragraphs have pointed out some of the numerical advantages of duality. Duality concepts also have application in nonlinear programming; however, so far these have been limited to SUMT methods involving convex functions. In this case, the duality theory provides a lower bound on the value of the unconstrained problem to test for convergence; see reference 24. Its use in nonlinear problems is not widespread.

LAGRANGE MULTIPLIERS AS A DESIGN AID

Another influence on choice of routine or method to solve an MP problem is postoptimal analysis.

When a constrained optimization problem is solved via sequential unconstrained minimization techniques (SUMT), additional information is available to the designer for a sensitivity analysis; i.e., postoptimal analysis. Suppose x^* is the solution to the following MP problem.

Minimize $f(x)$

subject to (46)

$$h_i(x) \leqslant b_i, \qquad i = 1, 2, \ldots, m$$

The engineer wishes information as to how $f(x^*)$ will change if b is changed a little; i.e., if the design requirements are changed, how will the performance be affected? The interesting result is:

$$\frac{\partial f(x^*(b))}{\partial b_i} = \lambda_i \tag{47}$$

where λ_i is a generalization of the classic Lagrange multiplier for finding the extrema with side conditions. The proof of (47) is contained in reference 44.

We recall from advanced calculus that an extrema problem with side conditions was: find the minimum (maximum) of $p(x_1, x_2, \ldots, x_n)$ subject to $q_1(x_1, \ldots, x_n) = \ldots = q_m(x_1, \ldots, x_n) = 0$. To solve this problem we

formed the Lagrangian $L(x_1, \ldots, x_n, \lambda_1, \ldots, \lambda_m) = p(x) + \sum_{i=1}^{m} \lambda_i q_i(x)$

then solved the (n+m) system of equations:

$$\frac{\partial L(x, \lambda)}{\partial x_i} = 0 \qquad i = 1, \ldots, n$$

$$q_i(x) = 0 \qquad i = 1, \ldots, m$$

for $(x_1, \ldots, x_n)$ with the λ_i's being introduced to help find the x_i. The Kuhn-Tucker theorem[45] allows us to define a meaningful and useful Lagrangian for inequality constraints.

We discuss the Kuhn-Tucker theorem to allow a generalization of Lagrange multiplier, and then discuss SUMT methods to illustrate obtaining the λ_i's numerically.

Rewriting the constraints to problem (46) as $g_i(x) = h_i(x) - b_i \leqslant 0$, we obtain problem (1):

Minimize $f(x)$

subject to (48)

$$g_i(x) \leqslant 0 \qquad i = 1, \ldots, m.$$

Then the Kuhn-Tucker theorem says essentially the following:

Let x^* be a solution to the above problem and assume the boundary of the constraint set has no "cusps;" then the following conditions hold.

1. There exist multipliers $\lambda_i \geqslant 0$, $i = 1, \ldots, m$ such that

$$\lambda_i g_i(x^*) = 0, \; i=1, 2, \ldots, m$$

2. $\nabla f(x^*) + \sum_{i=1}^{n} \lambda_i \nabla g_i(x^*) = 0$

The following example illustrates the Kuhn-Tucker condition:

Minimize $f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2$

subject to

$$g_1(x_1, x_2) = -x_2 + x_1^2 \leqslant 0$$

$$g_2(x_1, x_2) = -2 + x_1 + x_2 \leqslant 0$$

The global solution to the above is $(x_1^*, x^*_2) = (1, 1)$. The gradients at the optimum are

$$\nabla g_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \nabla g_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ and } \nabla f = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$

and the multipliers are $\lambda_1 = \lambda_2 = 2/3$. Finally we can write:

$$-\nabla f = +2/3\ \nabla g_1 + 2/3\ \nabla g_2.$$

Geometrically we have the following (fig. 6):

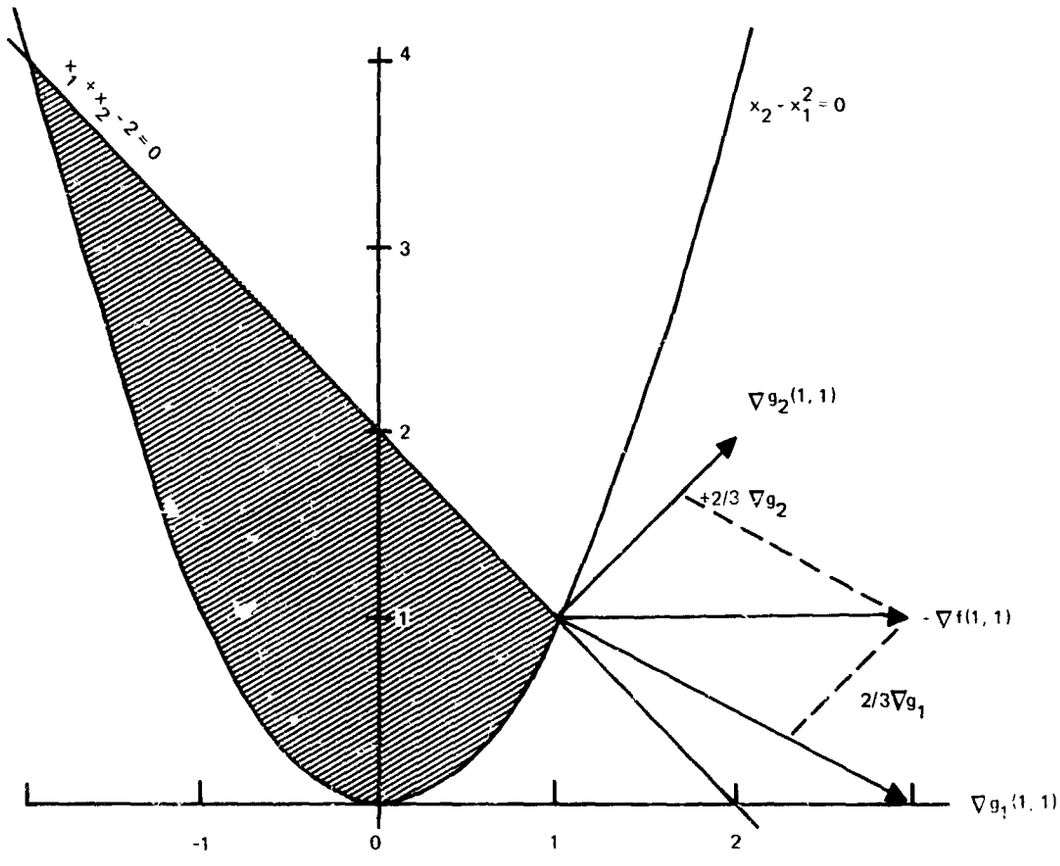


Figure 6. Kuhn-Tucker conditions.

Now that we have seen a need for knowing the Lagrange multipliers and have seen the geometrical interpretation, we turn to SUMT and the numerical calculation of the multipliers. The constrained optimization problem is transformed into an unconstrained one through the use of an auxiliary function, $\theta(x)$. $\theta(x)$ does one of two things, $\theta(x) \to \infty$ as $x \to 0$ or $\theta(x) = 0$ for x feasible and $\theta(x)$ positive for x infeasible. The three forms[44] of $\theta(x)$ we discuss are:

$$\text{Barrier: } B(x) = -\sum_{i=1}^{m} 1/g_i(x) \tag{49}$$

$$\text{Penalty: } P(x) = \sum_{i=1}^{m} (\text{Max } (g_i(x), 0))^{1+\epsilon} \tag{50}$$

$$\text{Logarithmic: } L(x) = -\sum_{i=1}^{m} \log(-g_i(x)) \tag{51}$$

SUMT works as follows: Using $\theta(x) = B(x)$, P(x) or L(x), we transform the constrained problem into a sequence of unconstrained problems, as in SUMT and Constraint Transformation.

$$\text{Minimize } D(x, r_k) = f(x) + p(r_k)\theta(x) \tag{52}$$

for $r_k > 0$. Then if x^* is the optimum for the constrained problem (48) and $x(r_k) = x_k$, the optimum for problem (52), it can be shown that $\lim_{k\to\infty} x_k = x^*$; i.e., $(r_k \to 0)$. Thus, we replace the constrained problem by a sequence of unconstrained problems.

In the barrier and logarithmic cases as $g_i(x) \to 0$ – i.e., x attempts to leave the constraint set – $g_i(x)$, must approach 0, and this causes B(x) and L(x) to increase rapidly. Since the routine wishes to minimize D(x, r), and we have an initial feasible point, intermediate x's are chosen feasible.

In the penalty function approach the test points are allowed to leave the constraint set; however, when they do so, a positive amount is added to the objective function. Again, since we seek to minimize D(x, r), points are selected within the constraint set. When B(x) or L(x) is used, we have an interior point method; and when P(x) is used, we have an exterior point method. With each type of $\theta(x)$ a slightly different technique is used to recover the λ_i.

Barrier Method:

Form the function

$$D(x, r) = f(x) + rB(x)$$

$$= f(x) + r \sum_{i=1}^{m} -1/g_i(x) \tag{53}$$

for $r > 0$. Let r_k be a monotonic decreasing sequence converging to zero. If x_k minimizes $D(x, r_k)$, then

$$0 = \nabla_x D(x_k, r_k) = \nabla f(x_k) + r_k \sum_{i=1}^{m} \frac{1}{g_i^2(x_k)} \nabla g_i(x_k) \tag{54}$$

Letting

$$\lambda_i^{(k)} = \frac{r_k}{g_i^2(x_k)} \tag{55}$$

then the Lagrangian becomes

$$L(x_k, \lambda_k) = f(x_k) + \sum_{i=1}^{m} \lambda_i^{(k)} \{-g_i(x_k)\} = f(x_k) + r_k \sum_{i=1}^{m} \frac{-1}{g_i(x_k)} \tag{56}$$

The multipliers can be computed by equation (55).

We note if $g_i(x^*) > 0$, then $\lambda_i^k = \frac{r_k}{g_i^2(x_k)}$ approaches 0 as $k \to \infty$, since

$r_k \to 0$.

If $g_i(x^*) = 0$, then by the Kuhn-Tucker theorem $\lambda_i \geq 0$; thus the limit $\frac{r_k}{g_i^2(x_k)}$

must be taken strictly as a quotient.

Logarithmic Penalty: Form $C(x, r) = f(x) - r \sum_{i=1}^{m} \log(-g_i(x))$ (57)

then at the optimum, x_k for $C(x, r_k)$ we have,

$$0 = \nabla_x C(x_k, r_k) = \nabla f(x_k) + r_k \sum_{i=1}^{m} \frac{-1}{g_i(x_k)} \nabla g_i(x_k) \tag{58}$$

Now, in a similar argument, let

$$\lambda_i^{(k)} = r_k / -g_i(x_k) \tag{59}$$

Since $(-g_i(x^*)) > 0$ in the feasible set, we have $\lambda_i^{(k)} > 0$. Thus, we can form the Lagrangian as before and equation (59) generates the multipliers.

We next give an example to illustrate this.
Lootsma[39] solves the following problem

$$\text{Minimize } f(x) = x_1^3 - 6x_1^2 + 11x_1 + x_3$$

subject to

$$g_1(x) = +x_1^2 + x_2^2 - x_3^2 \leqslant 0 \tag{60}$$

$$g_2(x) = -x_1^2 - x_2^2 - x_3^2 - 4 \leqslant 0$$

$$g_3(x) = +x_3 - 5 \leqslant 0$$

$$g_4(x) = -x_1 \leqslant 0$$

$$g_5(x) = -x_2 \leqslant 0$$

$$g_6(x) = -x_3 \leqslant 0$$

The optimum is $x^* = (0, \sqrt{2}, \sqrt{2})$ with $g_1(x^*) = g_2(x^*) = g_4(x^*) = 0$. The theoretical multipliers are

$$\lambda_1 = \lambda_2 = \sqrt{2/8}, = 0.1767766$$

$$\lambda_4 = 11; \lambda_3 = \lambda_5 = \lambda_6 = 0$$

For $r_k = 10^{-6}$, the numerical multipliers are

$$\lambda_1^{(k)} = \frac{r_k}{-g_1(x_k)} = \frac{10^{-6}}{5.66046\ 10^{-6}} = 0.176664$$

$$\lambda_2^{(k)} = \frac{r_k}{-g_2(x_k)} = \frac{10^{-6}}{6.65324\ 10^{-6}} = 0.15030$$

$$\lambda_4^{(k)} = \frac{r_k}{-g_4(x_k)} = \frac{10^{-6}}{9.2007\ 10^{-8}} = 10.8687$$

$$x_k = (9.2077\ 10^{-8}, 1.4121, 1.41422)$$

The final example is the Penalty Function method.

Let

$$C(x, r) = f(x) + \frac{1}{r} P(x),$$

where

$$P(x) = \sum_{i=1}^{m} (\max.\ (g_i(x), 0))^{1+\epsilon} \qquad \epsilon > 0.$$

Then at the optimum,

$$\nabla_x C(x_k, r_k) = 0 \text{ which implies } \nabla f(x) + \frac{1}{r} \nabla P(x) = 0 \qquad (61)$$

where $\nabla P(x) = +\Sigma(1+\epsilon)\,[\max.\ (g_i(x), 0)]^{\epsilon}\ \nabla g_i(x)$

If we let $\lambda_i^{(k)} = \frac{(1+\epsilon)}{r_k} \left|\max.[g_i(x), 0]\right|^{\epsilon}$

then $\lambda_i^{(k)} > 0$, since we have an exterior point method.

Thus, (61) becomes $\nabla f(x) + \Sigma\lambda_i \nabla g_i(x) = 0$,

and, letting x_k denote the optimum for r_k, we have

$$\lambda_i = \lim_{k\to\infty} \lambda_i^{(k)} = \lim_{k\to\infty} \left\{ \frac{1+\epsilon}{r_k} (\max.\ [g_i(x_k), 0])^{\epsilon} \right\}$$

In this way we can obtain the λ_i's.

In solving the Lootsma problem by the penalty method with $\epsilon = 1$, we obtained the following results for $r_k = .1*10^{-4}$:

$$x_k = (.1443*10^{-16},\ 1.4142,\ \ldots,\ 1.4141,\ \ldots)$$

$$g_1(x_k) = .8797110*10^{-6};\ g_2(x_k) = 0.8797477*10^{-6}$$

$$:_4(x_k) = 0.388414*10\text{-}14$$

$$\lambda_1^{(k)} = \frac{(1+\epsilon)}{r_k} (\max.\ (g_1, 0))^{\epsilon} = 2\left\{\frac{.879711*10^{-6}}{1.0\ *10^{-5}}\right\} = 0.1759422$$

$$\lambda_2^{(k)} = \frac{(1+\epsilon)}{r_k} (\max.\ (g_2, 0))^{\epsilon} = 2\left\{\frac{.87974779*10^{-6}}{1.0\ *10^{-5}}\right\} = 0.1759495580$$

$$\lambda_4^{(k)} = \frac{(1+\epsilon)}{r_k} (\max.\ (g_4, 0))^{\epsilon} = 2\left\{\frac{0.388414*10^{-14}}{10^{-5}}\right\} = 0.7768*10^{-11} \neq 11$$

The last four constraints of problem (60) were transformed by the following equation and not included as penalties:

$$x_1 = y_1^2$$

$$x_2 = y_2^2$$

$$x_3 = 5 \sin^2 (y(3)).$$

When these transformations are used, $\lambda_4^{(k)}$ cannot be obtained by the above formula.

INITIAL POINT AND SCALING

Two final topics in solution strategy depend a great deal on the specific problem: initial point and scaling of variables. If the constraints are numerous and complicated, then finding an initial feasible point can be an additional problem in itself.

Zoutendijk[47] gives a "simple" trick for a transformation which replaces the original problem by an equivalent problem for which an initial feasible point can readily be found. Suppose we are given:

Minimize f(x)

subject to

$$g_i(x) \leqslant 0 \qquad i = 1, 2, \ldots, m$$

and no initial point x° can be found by inspection or engineering knowledge such that $g_i(x^\circ) \leqslant 0$ for $i = 1, 2, \ldots, m$. We now form the following problem:

Minimize $f(x) + \mu\xi$

subject to $\quad g_i(x) - \rho_i\xi \leqslant 0$

$$\rho_i \geqslant 0 \qquad i = 1, 2, \ldots, m$$

where μ is a large number, ξ an additional variable, and $\rho_i = 1$ if $g_i(x^\circ) > 0$ and $\rho_i = 0$ if $g_i(x^\circ) \leqslant 0$. If $\xi \geqslant \max.\left\{g_i(x^\circ) \text{ for all } i \text{ such that } g_i(x^\circ) > 0\right\}$, then the point (x°, ξ) is feasible for the modified problem. In reference 17 Zoutendijk proves that, for μ sufficiently large, the modified problem will

have the same solution as the original. Note that in the attempt to minimize $f(x) + \mu\xi$, ξ is driven to zero, but to satisfy the constraints x must be simultaneously chosen feasible (for the original problem). This is the part of mathematical programming in which the user's previous engineering and design experience pays off, since the better the initial point, the faster the routine will converge.

Proper scaling of variables is still an art. Most computer routines for solving mathematical programming problems are designed to follow steep curved valleys and sharp ridges in locating an optimum. For a one-shot problem, scaling the variables is not worth the time; however, if a code is to be written to solve a large class of similar problems, then scaling may be worthwhile in long-term savings of machine time. We give an example, Pierre[48] which transforms a poorly shaped objective function into a "nice" bowl-shaped function which can quickly be minimized. Unfortunately, scaling is somewhat limited to unconstrained problems; in a constrained problem attempting to make the objective function easy to minimize, the scaling might destroy any useful properties the constraints enjoy.

Example:

If $f(x_1, x_2) = x_1^2 + 10x_1x_2 + 100x_2^2$, then f has a narrow valley and a minimum at $x_1 = x_2 = 0$. To transform f into a more desirable shape, we eliminate the cross product $10x_1x_2$ by letting $x_1 = z_1 - abz_2$ and $x_2 = bz_2$. Then f becomes

$$f = z_1^2 + (-2ab + 10b)\, z_1z_2 + (-10ab^2 + a^2b^2 + 100b^2)z_2^2 .$$

Letting $a = 5$ makes $(-2ab + 10b) = 0$ and $b = (1/75)^{1/2}$ solves $-50b^2 + 100b^2 + 25b^2 = 1$; thus, $f = z_1^2 + z_2^2$, which is easily minimized by almost any routine.

It is highly recommended, when the variables are bounded or the range of the variable is known, that the variable be normalized. For example, if the variable x_i lies in the range:

$$L_i \leqslant x_i \leqslant U_i,$$

then an appropriate normalization is given by:

$$0 \leqslant y_i \leqslant 1.0$$

where

$$y_i = \frac{x_i - L_i}{U_i - L_i}$$

Normalization is particularly useful in direct search algorithms for which we must arbitrarily choose a step size search increment, Δ. It can readily be 1% of 1.0 (0.01), which allows each variable to be changed proportionally, or any other realistic choice depending on our knowledge of the problem.

In summary, scaling an objective function is generally not practical, and we must rely on the properties of the algorithms to find the optimum; however, normalization is recommended wherever possible.

SUMMARY

Mathematical programming is a broad subject with many varied applications. Not all topics in MP are applicable to engineering design. We have tried to delineate those areas which are useful and the corresponding capabilities at NELC. The computer codes described have proved reliable on a wide range of problems; however, MP is an expanding area, and the new algorithms constantly being developed could render some of these codes obsolete. The applications-oriented user should not accept this list as complete or give up if his particular problem does not match an available routine. Perhaps a literature search will yield the appropriate method. The high-speed digital computer has provided an impetus to develop algorithms to solve MP problems which previously were too large to be handled. It is recommended that a continued effort be maintained to keep NELC up to date in the area of solving MP problems numerically. Logically, this should be a function of either a Center-wide computer users' group or of Computer Sciences Department. Other installations maintain a library of computer codes readily available to users; NELC should do the same.

In the applications area, mathematical programming has been a proved design tool. In reference 1, it was shown to be applicable to typical NELC problems. The second part of this report discussed some topics which were useful in obtaining actual solutions to design-related MP problems. The techniques have been used with routines available at NELC. We have tried to show that MP can be an aid to the design engineer and not his replacement. In fact to use MP effectively requires that the engineer be skillful in his field and be able to generate an accurate mathematical model of his design problem. We further recommend that a short course or continuing seminar be offered to NELC personnel to familiarize them with MP techniques. Such a course was given in-house in the fall of 1968 and was well received.

APPENDIX 1: APPLICATIONS OF INTEGER PROGRAMMING TO ENGINEERING DESIGN

In this appendix we report on the results of a literature search to determine the feasibility of using integer programming (IP) as a practical design aid in an ongoing NELC task – BAMS (Benchmarks for Applications of Microelectronics to Systems). The results were disappointing. Meaningful applications are still in the experimental stage, with results limited to relatively small test problems. The main hindrance to successful applications is not formulational difficulties, but the lack of reliable computer codes for solving the resulting IP problems. General-purpose IP codes are severely limited in the size problem they can solve (a maximum of 60 constraints and 60 variables at NELC), and they are sometimes unreliable. The majority of the applications of IP have been in the business world, and many algorithms for solving special IP problems have been developed; e.g., aircraft crew scheduling and warehouse placement. These methods rely on the special structure of the IP problem under consideration and have worked well. If the design engineer is lucky, his IP problem may fit one of these special methods (it is still an art to match the computer code to the posed IP problem); otherwise he must rely on the general IP codes. Here, application is ahead of theory.

We give a brief example which illustrates the computational difficulties of IP, a review of the state of the art of IP as related to BAMS, and finally a detailed example in which IP is used to solve a backboard wiring problem.

Most of the IP work has been done in the linear case; i.e.,

$$\text{Minimize } f(x) = c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

$$\text{subject to } g_1(x) = a_{11}x_1 + a_{12}x_{12} + \ldots + a_{1n}x_{1n} \leqslant b_1$$

$$\vdots$$

$$g_m(x) = a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leqslant b_m$$

$$x_i \geqslant 0 \text{ and } x_i \text{ an integer}$$

Thus, the integer linear programming (ILP) problem is just the LP problem with the additional constraint that the solution be integral. Gomory[49] has developed an algorithm which theoretically solves the above problem in a finite number of steps N; however, N can be a large number and hence impractical for some problems. Since this area of mathematics is so useful, many methods (some heuristic in nature) for solving special ILP problems have been reported which work well. One method, which appears obvious, is to solve the associated LP problem, then round to the nearest (in some sense) integer-valued vector and use that for the solution. Many times this

will work fine, if it is not too critical to have the optimum. If it is necessary to have the optimum and know that it is the optimum, then other methods must be used. The following example points out some difficulties which can arise.

$$\text{Minimize } z = f(x_1, x_2) = -x_1 - 4x_2$$

subject to $0 \leq x_1 \leq 4.55$

$0 \leq x_2 \leq 4.0$

$.5x_1 + x_2 \leq 5.2$ x_1, x_2 integers

The constraint set (denoted by +) looks as follows (fig. 7):

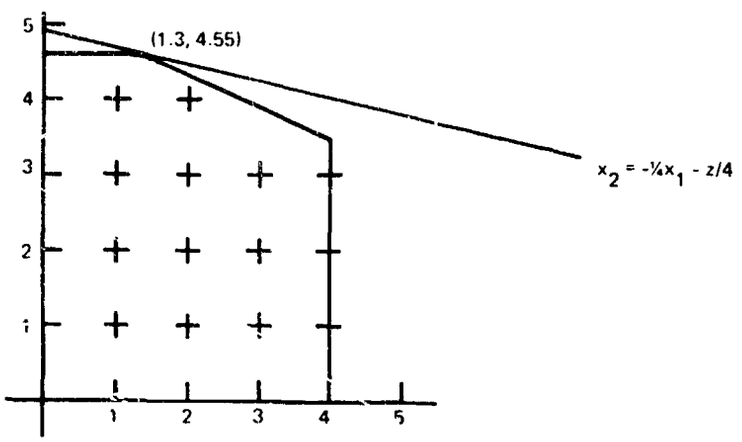


Figure 7. Integer constraint set.

When the associated LP problem is solved (by inspection), we obtain (1.3, 4.55). Rounding to the nearest vector with integer components yields (1,5), but (1,5) is not feasible. If we take the closest feasible point to (1.3, 4.55) in the Euclidean norm sense, then we have (1,4) for a solution, which yields $z = f(1,4) = -17$. However, another feasible point (2,4) yields $z = f(2,4) = -18$ and this point is the solution. As the number of variables and constraints increases, it becomes difficult to find a feasible point to the ILP near the LP optimum.

The field of digital systems design has made the most engineering use of IP, but again designers have had only limited success. Logic designers have used ILP as a theoretical tool and have solved small problems numerically. Muroga[51] perhaps has the most recent application (see references 52 and 53 also). He discusses designing optimal networks of the "feedforward" type by IP and defines a generalized gate called a threshold gate. His IP formulation allows a wide choice of objective functions, depending on the application,

as well as the inclusion of any design constraints. If R is the number of gates in the design, then his associated ILP problem has R^2 variables. For R of any typical size in a logic design problem, the resulting ILP problem would be intractable. Presently, work in this area is of academic interest only. Also, with the abundance of off-the-shelf MSI and LSI components, little design is done at the gate level (except by those manufacturing the aforementioned items). Thus, the extra effort to formulate the logic design problem as an ILP does not appear to be cost-effective.

The next pertinent application area is that of actual component and circuit layout. Kodres[54] develops the theory for solving the circuit layout problem, which he defines as follows:

"The circuit layout problem is viewed as a sequence of four subproblems.

1. The determination of standard replaceable modules.

2. The partitioning of circuits into groups subject to input-output restrictions.

3. The selection of replaceable modules.

4. The circuit placement and the interconnection problem."

Kodres uses graph theory, combinatorics, and integer programming to formulate the problem. The actual casting of parts 3 and 4 as an IP problem is partially in terms of some graph theoretical concepts and requires more background than we can present here. This paper points the way for future work in this area; again, the ideas are far ahead of practical methods for implementing them. Breuer in reference 55 poses part 4 as a single ILP problem, in straightforward terms, which we outline in the following paragraphs. Other applications are in coding theory and satellite communications network design; see references 56 - 58.

BREUER'S PLACEMENT AND INTERCONNECTION IP FORMULATION

The backboard wiring problem consists of three subproblems: the placement problem, the connection problem, and the routing and installation problem; each is dependent on the other two. We discuss the first two and pose them as a single ILP problem which, when solved, will simultaneously solve both problems.

PLACEMENT PROBLEM

Given B objects, connect each object to a subset of the remaining (B-1) objects. The objects are constrained to lie on grid points which represent the backboard of a computer, or any digital system hookup. The object is to place all the modules so that the hookup wire is of minimal length.

CONNECTION PROBLEM

Given S fixed objects which are to be made electrically common, connect the objects so that the total interconnection length is minimal.

INTEGER PROGRAMMING FORMULATION

Given B objects to be placed at the intersections of the rectangular grid (fig. 8).

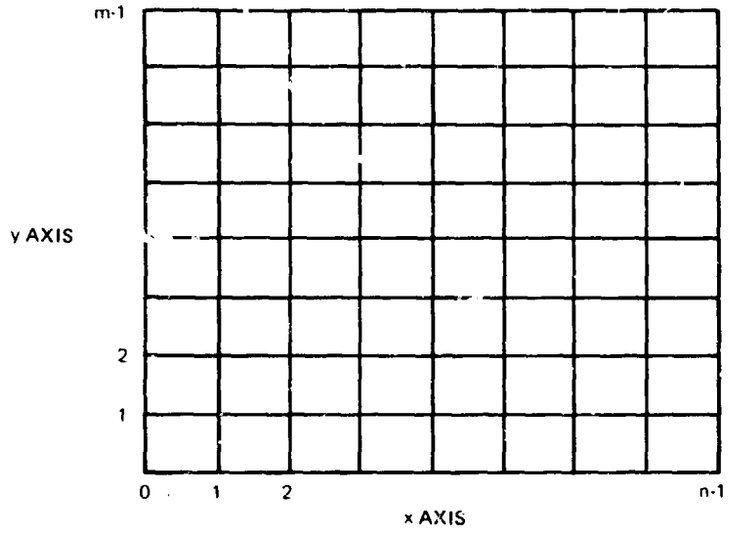


Figure 8. Back plane grid.

Let x_i, $i = 0,1, \ldots, n-1$ be the x coordinate of the ith object

y_i, $i = 0,1, \ldots, m-1$ be the y coordinate of the ith object

and $B \leqslant mn$.

The interconnection distance between the ith and jth object is defined to be

$$d_{ij} = |x_i - x_j| + k\,|y_i - y_j|.$$

We note that no two objects can occupy the same spot at the same time. Also given is a list of the desired connections. We wish to uniquely position the set of B objects at the intersections and determine which objects should be directly connected together, in a manner such that total interconnection distance of those objects, directly connected, is minimal. The hard parts are making sure no two objects occupy the same spot and getting a linear relation for d_{ij}.

The constraint that no two objects lie at the same point requires that if $x_i = x_j$, then $y_i \neq y_j$ for $i \neq j$; or if $y_i = y_j$, then $x_i \neq x_j$ for $i \neq j$.

We list the required constraints for the placement problem and then explain how they meet the conditions of the problem.

$$x_i - x_j \leqslant n\,\delta_{ij} \tag{A1}$$

$$x_j - x_i + 1 \leqslant n(1 - \delta_{ij}) \tag{A2}$$

$$\delta_{ij} \leqslant 1 \quad \text{i.e. } \delta_{ij} = 0 \text{ or } 1 \tag{A3}$$

$$\alpha_{ij} \leqslant n\,\delta_{ij} \tag{A4}$$

$$x_i - x_j \leqslant \alpha_{ij} \leqslant x_i - x_j + n(1 - \delta_{ij}) \tag{A5}$$

$$\alpha_{ji} \leqslant n(1 - \delta_{ij}) \tag{A6}$$

$$x_j - x_i \leqslant \alpha_{ji} \leqslant x_j - x_i + n\,\delta_{ij} \tag{A7}$$

for all $i > j$, $j = 1,2, \ldots, B-1$

From inequalities (A1) and (A2) we find that if $x_i > x_j$, then $\delta_{ij} = 1$ and $\delta_{ij} = 0$, if $x_i \not> x_j$. Inequalities (A4) - (A7) give a representation for $|x_i - x_j|$ as follows. Since (A4) and (A5) yield $0 < \alpha_{ij} = x_i - x_j \leqslant n - 1$, if $x_i > x_j$ and $\alpha_{ij} = 0$ otherwise. In a similar fashion $0 < \alpha_{ji} = x_j - x_i$, if $x_j > x_i$; thus, we have $\alpha_{ij} + \alpha_{ji} = |x_i - x_j|$.

Now let n and β play analogous roles for y; then $\beta_{ij} + \beta_{ji} = |y_i - y_j|$. Thus, d_{ij} can be stated in terms of these auxiliary variables as

$$d_{ij} = \alpha_{ij} + \alpha_{ji} + k(\beta_{ij} + \beta_{ji})$$

Some additional constraints guarantee realizability.

$$\alpha_{ij} + \alpha_{ji} + k(\beta_{ij} + \beta_{ji}) \geqslant 1 \tag{A8}$$

$$x_i \leqslant n-1 \quad i = 1, 2, \ldots, B \tag{A9}$$

$$y_i \leqslant m-1 \quad i = 1, 2, \ldots, B \tag{A10}$$

Inequality (A8) guarantees that two objects do not occupy the same grid intersection. All x_i and y_i are nonnegative integers. Note that the above inequalities do not depend on how the objects are connected, but only guarantee that specific necessary conditions will be satisfied. Inequalities (A1) through (A10) represent the constraints for the placement problem. We now turn to the problem of optimally interconnecting the B objects.

Assume that there are C independent circuits where the jth circuit can be connected in P_j different acceptable ways. Let $f_{ij}(\mathbf{d})$ be an expression for the total length of wire in the ith way of connecting the jth circuit. For example, if the sixth circuit consists of three objects (1, 2, 3) and the first

may have only one connection to it, then

$$f_{16} = d_{21} + d_{32} \quad \text{and} \quad f_{26} = d_{31} + d_{32}$$

The two final constraints which relate to the connection problem are:

$$V_j \geq f_{ij}(d) + (\gamma_{ij} - 1)Q \tag{A11}$$

and

$$\sum_{i=1}^{P_j} \gamma_{ij} = 1 \quad \text{for } j = 1 \ldots C \tag{A12}$$

Equation (A12) implies that each circuit is connected and γ_{ij} is 0 or 1. In (A11) $Q = \max. f_{ij}(d)$ for $i = 1, 2, \ldots, P_j; j = 1, 2, \ldots, C$. Then the resulting objective function is

$$\min. Z = \sum_{j=1}^{C} V_j$$

The objective function and inequalities (A1) - (A11) form the ILP representing the combined placement and connection problem. Although the formulation seems straightforward, the resulting ILP problem can be large, even for small values of B. Breuer gives the following relations between the number of variables I, the number of constraints W, and B for the placement problem; i.e., constraints (A1) - (A10).

$$I(B) = (B/2)(19B-15)$$

$$W(B) = B(3B-1)$$

Thus, for B=5 (which could be manually positioned quickly and most likely optimally), we have I(5) = 200, W(5) = 70; for B = 10 (still not too large), I(10) = 875, W(10) = 290. These ILP problems are very large and are beyond the capabilities of today's methods.

It seems that, unless newer formulational techniques are developed which lead to smaller IP problems, or the computational capabilities solving IP increase rapidly, this approach will remain a theoretical tool with no practical applications. The following quote (Glover[59], p. 1) sums up the current state of affairs in integer programming: "Since its inception integer linear programming has, paradoxically, been a source of both promise and disappointment. Promise because there are manifold and compelling opportunities for its application; disappointment because it has made only the most dubious progress in spite of these opportunities."

APPENDIX 2: USER INFORMATION

H1 – H2 OPTALG

CATALOG IDENTIFICATION:

H1 – H2 OPTALG

PROGRAMMER:

F. S. Hillier, Stanford University, adapted for NELC by D. Klamer, Decision and Control Technology Division.

PURPOSE:

An algorithm for solving the pure integer linear programming problem.

$$\text{Maximize } x_o = \sum_{j=1}^{n} c_j x_j$$

subject to

$$\text{(i)} \quad \sum_{j=1}^{n} a_{ij} x_j \leqslant b_j \qquad (i=1,2,\ldots,m)$$

$$\text{(ii)} \quad x_j \geqslant 0 \qquad (j=1,2,\ldots,n)$$

$$\text{(iii)} \quad x_j \text{ is an integer} \qquad (j=1,2,\ldots,n)$$

RESTRICTIONS AND LIMITATIONS:

The dimensions of the constraint matrix $A(I,J) = a_{ij}$ have to be less than or equal to 61 X 61; i.e., $m \leqslant 61$ and $n \leqslant 61$.

LANGUAGE:

FORTRAN IV

COMPUTER CONFIGURATION:

Go step REGION = 336k
IBM 360/65

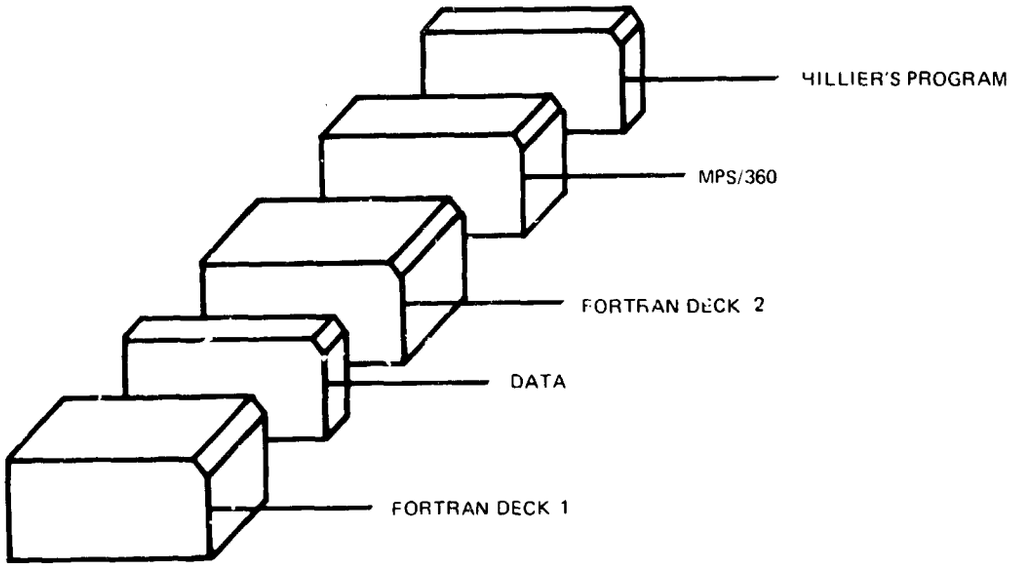
METHOD:

An initial noninteger solution must be obtained from the related linear programming problem (i.e., x_j is not necessarily an integer) as well as the resulting basis inverse. To accomplish this, we have chosen the linear programming routine MPS/360.[18] Data are taken directly from MPS/360 and input directly into Hillier's program,[31] without user intervention, in one multistep computer run.

The advantage of using this modified version of Hillier's program is the time saved from obtaining the basis inverse and optimal solution; the time

spent for punching these input cards is also saved. The data needed are exactly the same as the first three card groups of Hillier's program. These are:

Card group 1	Any alphanumeric characters to identify the problem in 20A4 Format
Card group 2	m, n, KL in 315 Format where A(I,J) is of size mxm and KL = 1
Card group 3	the arrays A, b, c in 15 F5.0 Format. A(I,J) is the constraint matrix, B(I) is the right-hand side, and C(J) is the objective function. The A is read in one row at a time.



FORTRAN DECK 1	Setup data for MPS/360.
DATA	Constraint matrix, right-hand side, and objective function.
FORTRAN DECK 2	Using READCOM from MPS/360 obtains basis inverse, JPM(I), optimal solution, and starting integer solution. Stores information on disk.
MPS/360	Computes basis inverse and optimal noninteger solution.
HILLIER'S PROGRAM	Computes the optimal integer solution.

FORTRAN DECK 1

This deck sets up the data for MPS/360, since the format for MPS/360 is long and cumbersome. The constraint matrix A(I,J) is normalized, as is the right-hand side B(I); this is done one row at a time. The subroutine XPUNCH places the data into a disk file in proper format for MPS/360, from which MPS/360 reads the data. A printout is given of the data that are placed on disk. (Note: This data set is placed into a disk file called FT01F001. The data are in normalized form, and, since MPS/360 is designed to find the minimum, the signs of the cost coefficients (objective function) are changed.) The constraint matrix A(I,J), the right-hand side B(I), and the cost coefficients C(J) are also stored on the disk file called FT02F001.

FORTRAN DECK 2 (DATAHILL)

This deck is a temporary update that is concatenated onto MPS/360, under the name DATAHILL. It uses READCOMM,[20] which is a subroutine designed to augment MPS/360 with procedures written in FORTRAN language. DATAHILL retrieves from MPS/360 the basis inverse, the order of the basic variables, the optimal noninteger solution, and a starting feasible solution to the integer linear programming problem. These data are then added to the data from the first FORTRAN deck on the disk file FT02F001.

The starting feasible solution is a lower bound on the value of the objective function. To obtain this feasible solution, we have chosen to do the following: If the cost coefficient is positive, round the corresponding variable of the optimal noninteger solution down to the next largest integer. If the cost coefficient is negative, round up to the next smallest integer. (Note: The cost coefficients are placed into the disk file called FT03F001 in the first FORTRAN deck and are read by DATAHILL; the constraint matrix and right-hand side are also passed.) This rounded solution is checked for feasibility. If the solution is feasible, then proceed to the next step. If the solution is not feasible, then try a rounding procedure to satisfy the constraint violated. (User may set the maximum number of iterations or changes in the solution.) In order for Hillier's program to be executed, a feasible integer solution must be found; if such a solution is not found, then Hillier's program is skipped and all the data are punched out on cards. This information includes the name, constraint matrix A(I,J), right-hand side B(I), cost coefficients C(J), the basis inverse, the order of the basic variables, and the optimal noninteger solution. The user may then supply his own starting integer solution and run the problem directly from LLOAD, using the punched data produced in the last step. (See Part II.)

If a feasible integer solution is obtained and the user wishes to have the above data also, then either of two methods may be used. First, there

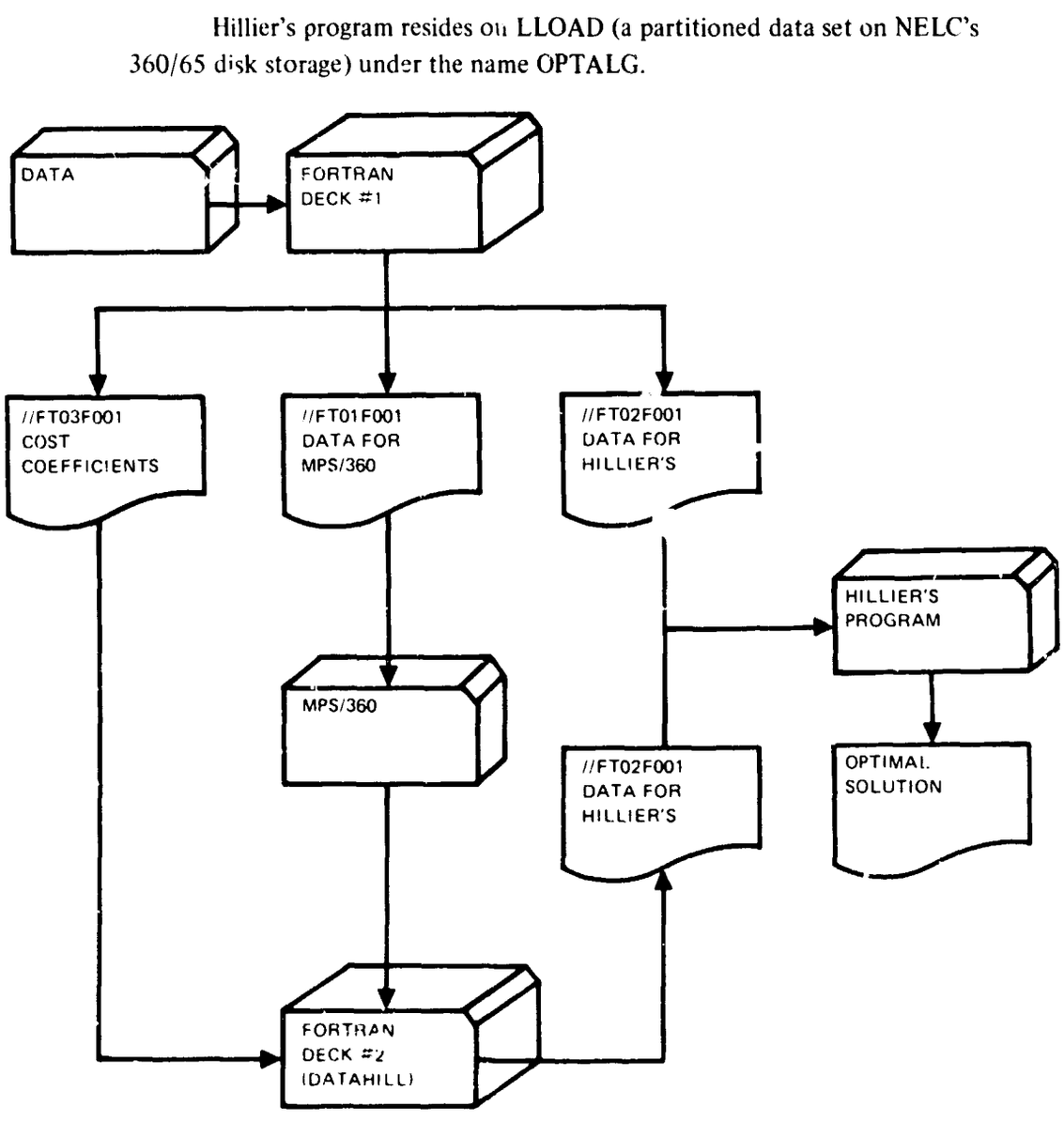
are comment cards in the program to punch out each of the groups of data. All that is required is to remove the "C" from the cards in the program corresponding to which groups are to be punched out. The second method requires two changes in the JCL cards. See the JCL listing at the beginning of the program.

MPS/360

MPS/360 is an IBM supplied application program, "Mathematical Programming System/360." MPS/360 obtains an optimal solution (non-integer) from the related linear programming problem and finds the inverse of the basis.

HILLIER'S PROGRAM

Hillier's program resides on LLOAD (a partitioned data set on NELC's 360/65 disk storage) under the name OPTALG.



H1-H2 OPTALG – Part II

This section covers the necessary input for Hillier's program when run without the adapted program to generate the data. The following is the necessary input:

Card input 1 — Any alphanumeric characters to identify the problem in 20A4 Format; e.g., "Thompson Number 8"

Card group 2 — m, n, KL in 315 Format ($m \leqslant 61$, $n \leqslant 61$) where m is the number of rows of the constraint matrix A(I,J) and n is the number of columns

$$KL = \begin{cases} 1 \text{ if the basis inverse is in} \\ \text{normalized form} \\ 0 \text{ otherwise} \end{cases}$$

Card group 3 — This group of cards contains the arrays $\underline{A}$, $\underline{b}$, $\underline{c}$; the Format is 15F5.0. A(I,J) is the constraint matrix, B(I) is the right-hand side, and C(J) is the row matrix of the cost coefficients (or objective function). The $\underline{A}$ array is read in one row at a time. (For example, if m=2, m=16, the cards would be:
$a_{1,j}$(j=1,2 . . . , 15) in the first 75 columns;
$a_{1,16}$ in the first 5 columns;
$a_{2,j}$(j=1,2, . . . , 15) in the first 75 columns;
$a_{2,16}$ in the first 5 columns;
b_1, b_2 in the first 10 columns;
c_j(j=1,2, . . . , 15) in the first 75 columns;
c_{16} in the first 5 columns.)

Card Group 4 — This group of cards contains the basis inverse in 6F13.5 Format. The rows are read in sequentially. In the example listing,

$$\begin{array}{cccc} BB_{1,1} & BB_{1,2} & BB_{1,3} & \cdots & BB_{1,6} \\ BB_{1,7} & BB_{1,8} & BB_{2,1} & \cdots & BB_{2,4} \\ BB_{2,5} & BB_{2,6} & BB_{2,7} & \cdots & BB_{3,2} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \end{array}$$

Card Group 5 — JPM(i), i=1,2 . . . , in 1514 Format, where JPM(i) is the index of the i^{th} basic variable (including slack variables) from the simplex code.

Card Group 6 — The optimal solution to the related linear programming problem, x(j), in 6F13.5 Format.

Card Group 7 A starting optimal solution to the Integer Linear Programming problem, XF(j), in 6F13.5 Format.

See the sample problem following. The correspondence of cards with the above card groups is as follows:

Card Group	Card in Sample Problem
1	1
2	2
3	3-12
4	13-23
5	24
6	25-26
7	27-28

(Note: The first three card groups are the same as the three card groups for the adapted version of Hillier's program.)

SAMPLE PROBLEM

```
//PRELIM  JOB (055369,6202044,F,D,5,5,1000),DKLAMER,
//          MSGLEVEL=1,CLASS=L
/ MESSAGE       000
//S1 EXEC FORTGCLG,TIME=1,REGION.GO=64K
//FORT.SYSIN DD *
      DOUBLE PRECISION IA(61),BUFFER(61)
      DIMENSION A(61,61), B(61), C(61), NAME(20), SUM(61)
574   FORMAT(20A4)
      READ(5,574)(NAME(I),I=1,20)
575   FORMAT(1H ,5X,20A4)
      WRITE(2,574)(NAME(I),I=1,20)
      WRITE(6,575)(NAME(I),I=1,20)
201   FORMAT(3I5)
      READ(5,201) M,N,KL
      WRITE(2,201)M,N,KL
200   FORMAT(15F5.0)
      WRITE(6,503)
      DO 183 I=1,M
      READ (5,2001)(IA(J),J=1,N)
      WRITE(2,2001)(IA(J),J=1,N)
      WRITE(3,2001)(IA(J),J=1,N)
      CALL CORE(BUFFER,488)
      WRITE(8,2002)(IA(J),J=1,N)
      CALL CORE(BUFFER,488)
      READ(8,2003) (A(I,J),J=1,N)
      WRITE(6,500) (A(I,J),J=1,N)
183   CONTINUE
      WRITE(6,504)
      READ (5,2001)(IA(J),J=1,M)
      WRITE(2,2001)(IA(J),J=1,M)
      WRITE(3,2001)(IA(J),J=1,M)
      CALL CORE(BUFFER,488)
      WRITE(8,2002)(IA(J),J=1,M)
      CALL CORE(BUFFER,488)
      READ(8,2003) (B(J),J=1,M)
      WRITE(6,500)(B(J),J=1,M)
      WRITE(6,505)
      READ (5,2001)(IA(J),J=1,N)
      WRITE(2,2001)(IA(J),J=1,N)
      WRITE(3,2001)(IA(J),J=1,N)
      CALL CORE(BUFFER,488)
      WRITE(8,2002)(IA(J),J=1,N)
      CALL CORE(BUFFER,488)
      READ(8,2003) (C(J),J=1,N)
      WRITE(6,500)(C(J),J=1,N)
2001  FORMAT(15A5)
2002  FORMAT(6A5)
2003  FORMAT(61F5.0)
500   FORMAT(1H ,15F8.2)
501   FORMAT(1H ,15F8.4)
555   FORMAT(1H ,10F12.5)
567   FORMAT(1H ,15F8.4)
```

```
503   FORMAT(28H CONSTRAINT MATRIX A(I,J) IS)
504   FORMAT(24H RIGHT HAND SIDE B(I) IS)
505   FORMAT(27H COST COEFFICIENTS C(J) ARE)
506   FORMAT(43H THE NORMALIZED CONSTRAINT MATRIX A(I,J) IS)
507   FORMAT(39H THE NORMALIZED RIGHT HAND SIDE B(I) IS)
      K=0
      DO 1111 I=1,M
      K=K+1
      SUM(K)=0.0
      DO 2222 J=1,N
      SUM(K)=SUM(K)+A(I,J)**2
2222  CONTINUE
      SUM(K)=SQRT(SUM(K))
1111  CONTINUE
      DO 3333 I=1,M
      DO 4444 J=1,N
      A(I,J)=A(I,J)/SUM(I)
4444  CONTINUE
3333  CONTINUE
      WRITE(6,506)
      DO 5555 I=1,M
5555  WRITE(6,567)(A(I,J),J=1,N)
      DO 7777 I=1,M
7777  B(I)=B(I)/SUM(I)
      WRITE(6,507)
      WRITE(6,501)(B(I),I=1,M)
508   FORMAT('1 THE FOLLOWING DATA HAS BEEN STORED ON DISK ')
      WRITE (6,508)
      CALL XPUNCH (A,B,C,M,N)
      STOP
      END


      SUBROUTINE XPUNCH (A,B,C,M,N)
      DIMENSION A(61,61), B(61), C(61)
      WRITE (1,111)
      PRINT 110
      WRITE (1,113)
      PRINT 112
      WRITE (1,115)
      PRINT 114
      DO 1 KROW = 1, M
      KROW10  =  KROW + 10
      WRITE (1,107) KROW10
      PRINT 106,KROW10
1     CONTINUE
      WRITE (1,109)
      PRINT 108
      M1      =  M+1
      DO 10 J =  1,N
      K       =   11
      I       =   0
```

```
5       CONTINUE
        K1      =  K
        K2      =  K+1
        L       =  J+10
        I       =   I + 1
        IF (K2 .GE. M1 + 10) GO TO 11
        WRITE (1,100) L,K1,A(I,J),K2,A(I+1,J)
        PRINT 100,L,K1,A(I,J),K2,A(I+1,J)
        K       =  K+2
        I       =   I + 1
        GO TO 5
11      IF (K .EQ. M1 +10) GO TO 12
         C(J)   =  - C(J)
        WRITE (1,101) L,K1,A(I,J),C(J)
        PRINT 101,L,K1,A(I,J),C(J)
        GO TO 10
12       C(J)   =  - C(J)
        WRITE (1,102) L,C(J)
        PRINT 102,L,C(J)
10      CONTINUE
        WRITE (1,105)
        PRINT 104
        DO 20 I = 1,M
        IR      =  I + 10
        WRITE (1,103) IR,B(I)
        PRINT 103,IR,B(I)
20      CONTINUE
        WRITE (1,117)
        PRINT 116
100     FORMAT(4X,'C',I2,7X,'R',I2,7X,F12.5,3X,'R',I2,7X,F12.5)
101     FORMAT(4X,'C',I2,7X,'R',I2,7X,F12.5,3X,'C',9X,F12.5)
102     FORMAT(4X,'C',I2,7X,'O',9X,F12.5)
103     FORMAT(4X,'CONST',5X,'R',I2,7X,F12.5)
104     FORMAT ( ' RHS' )
105     FORMAT ( 'RHS' )
106     FORMAT( '  L  R',I2)
107     FORMAT ( ' L  R',I2)
108     FORMAT ( ' COLUMNS')
109     FORMAT ( 'COLUMNS' )
110     FORMAT (' NAME          OPTBAS')
111     FORMAT ( 'NAME          OPTBAS')
112     FORMAT (' ROWS')
113     FORMAT ( 'ROWS')
114     FORMAT('  N  O')
115     FORMAT( ' N  O')
116     FORMAT (' ENDATA')
117     FORMAT ( 'ENDATA')
        RETURN
        END
```

```
//GO.FT01F001 DD UNIT=SYSDA,SPACE=(TRK,(20,10),RLSE),
// DISP=(,PASS),
// DCB=(RECFM=FBS,LRECL=80,BLKSIZE=880)
//GO.FT02F001 DD UNIT=SYSDA,SPACE=(TRK,(20,10)),
// DISP=(,PASS),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=880)
//GO.FT03F001 DD UNIT=SYSDA,SPACE=(TRK,(20,10),RLSE),
// DISP=(,PASS),
// DCB=(RECFM=FBS,LRECL=80,BLKSIZE=880)
//GO.SYSIN DD *



      $$$$$$
      $
      $ DATA
      $
      $$$$$$


//STEPONE  EXEC FORTGCL
//FORT.SYSLIN DD DISP=(NEW,PASS)
//FORT.SYSIN DD *
       INTEGER DIFF,FILE
       INTEGER*2 COMP(61),C,R
       DIMENSION BASE(61,61),NROW(61),NCOLM(61),JPM(61),IROW(61),
      1 BUFFER(61),CC(61),XF(61)
       DIMENSION A(61,61),B(61),
      1    IDX1(61),IDXF(61),IDX2(61),XFMIN(61)
       REAL*8 NAME,XLIST(30), OUT1(62),OUT2(3721),OUT(61)
       DATA BASE/3721*0.0/
       LK = 0
       FILE=4
C
C  TO OBTAIN THE NUMBER OF ROWS AND COLUMNS OF THE
C    CONSTRAINT MATRIX(INCLUDING THE OBJECTIVE FUNCTION)
C
       CALL ARRAY(FILE,INDIC,NAME)
       CALL VECTOR(FILE,INDIC,XLIST)
       MROWS  =IFIX(SNGL(XLIST(9)))
       NCOLMN=IFIX(SNGL(XLIST(10)))
       MROWS1 = MROWS- 1
       MN1=MROWS1 * NCOLMN
       MN=NCOLMN*MROWS
C
C  TO OBTAIN THE BASIS
C
       CALL ARRAY(FILE,INDIC,NAME)
       DO 10 J=1,MN
       CALL VECTOR(FILE,INDIC,XLIST)
       IF(J.GT.MROWS) GOTO 12
       OUT1(J) = XLIST(1)
12     IF(MOD(J,MROWS )-1) 11 ,115,11
```

```
11      LK =LK + 1
        OUT2(LK) = XLIST(2)
115     IF(INDIC-1)  20,20,10
10      CONTINUE
20      CONTINUE
C
C   TO OBTAIN THE OPTIMAL SOLUTION
C
        CALL ARRAY(FILE,INDIC,NAME)
        CALL VECTOR(FILE,INDIC,XLIST)
        CALL ARRAY(FILE,INDIC,NAME)
        CALL VECTOR(FILE,INDIC,XLIST)
        CALL ARRAY(FILE,INDIC,NAME)
        DO 201 J=1,NCOLMN
        CALL VECTOR(FILE,INDIC,XLIST)
        OUT(J)=XLIST(3)
        IF(INDIC-1)  200,200,201
201     CONTINUE
C
C   TO PRINT OUT THE OPTIMAL SOLUTION
C
101     FORMAT(6F13.5)
102     FORMAT(1X,6F13.5)
103     FORMAT(' THE OPTIMAL NONINTEGER SOLUTION IS '//)
        WRITE(6,103)
        WRITE(6,102) (OUT(J),J=1,NCOLMN)
C
C   TO DETERMINE WHICH VARIABLES ARE ACTIVE
C
        CALL CORE(BUFFER,240)
        WRITE(8,104) (OUT1(J),J=2,MROWS)
104     FORMAT(61A4)
        CALL CORE(BUFFER,200)
        READ (8  ,105) (COMP(J),IROW(J),J=1,MROWS1)
105     FORMAT(61(1A1,I2,1X))
        DATA R/'R '/,C/'C '/
        NOROW  = 0
        NOCOLM = 0
        DO 500 J=1,MROWS1
        IF(COMP(J).EQ.R) NOROW  = NOROW  + 1
500     IF(COMP(J).EQ.C) NOCOLM = NOCOLM + 1
        IF((NOROW.EQ. 0 ).AND.(NOCOLM.EQ. 0 )) GOTO 100
        IF(NOROW.EQ. 0 ) GOTO 600
        IF(NOCOLM.EQ. 0) GOTO 650
C
C    TO COMPUTE THE JTH BASIC VARIABLE FOR OPTALG USING MPS/360
    INFORMATI
C
        DO 56 J=1,NOROW
56      NROW(J) = IROW(J)
        I = 0
        NTOT = NOROW + NOCOLM
```

```
      N1 = NUROW + 1
      DO 57 J=N1,NTOT
      I = I+ 1
57    NCOLM(I) = IROW(J)
      GOTO 150
C
C  BASIS INVERSE IS BASIS
C
600   DO 601 J=1,NOCOLM
601   JPM(J) = J
      WRITE(6,109)
      DO 602 I=1,MROWS1
602   WRITE(6,101)  (OUT2(I+(J-1)*MROWS1),J=1,NCOLMN)
      WRITE(2,101) ((OUT2(I+(J-1)*MROWS1),J=1,NCOLMN),I=1,MROWS1)
C
C  IF YOU WANT THE BASIS INVERSE PUNCHED OUT
C     (THERE ARE THREE (3) CARDS FOR THE BASIS INVERSE)
C     REMOVE THE 'C' FROM THE FOLLOWING CARD.
C     WRITE(7,101) ((OUT2(I+(J-1)*MROWS1),J=1,NCOLMN),I=1,MROWS1)
C
      GOTO 202
C
C  BASIS INVERSE IS IDENTITY MATRIX
C
650   DO 651 J=1,NOROW
      JPM(J) = MROWS1 + J
651   BASE(J,J) =1.0
      WRITE(6,109)
      WRITE(6,101) ((BASE(I,J),J=1,MROWS1),I=1,MROWS1)
      WRITE(2,101) ((BASE(I,J),J=1,MROWS1),I=1,MROWS1)
C
C  IF YOU WANT THE BASIS INVERSE PUNCHED OUT
C     (THERE ARE THREE (3) CARDS FOR THE BASIS INVERSE)
C     REMOVE THE 'C' FROM THE FOLLOWING CARD.
C     WRITE(7,101) ((BASE(I,J),J=1,MROWS1),I=1,MROWS1)
C
      GOTO 202

150   CONTINUE
      L=1
      KL=1
      NORM1 = NOROW - 1
      IF(NROW(1)-11) 100,30,22
30    JPM(1) = NROW(1) + NCOLMN - 10
      L = L+1
      GOTO 47
22    DIFF = NROW(1) - 11
      DO 46 K=1,DIFF
      JPM(L) = NCOLM(KL) - 10
      KL = KL+1
      L = L+1
46    CONTINUE
      JPM(L) = NROW(1) + NCOLMN - 10
      L = L + 1
47    CONTINUE
70    DO 50 I=1,NORM1
```

```
      DIFF = NROW(I+1) - (NROW(I)+1)
      IF (DIFF) 100,35,40
35    JPM(L) = NROW(I+1) + NCOLMN - 10
      L = L+1
      GOTO 50
40    DO 45 K=1,DIFF
      JPM(L) = NCOLM(KL) - 10
      KL = KL+1
      L = L+1
45    CONTINUE
      JPM(L) = NROW(I+1) + NCOLMN - 10
      L = L+1
50    CONTINUE
      DIFF = 10 + NTOT - NROW(NOROW)
      IF(DIFF) 100,55,60
55    JPM(NTOT) = NROW(NOROW) + NCOLMN - 10
      GOTO 65
60    DO 65 K=1,DIFF
      JPM(L) = NCOLM(KL) - 10
      KL = KL+1
      L = L+1
65    CONTINUE
C
C  TO OBTAIN THE BASIS INVERSE
C
      J1 = 0
      J2 = 1
      DO 300 JK=1,NTOT
      IF(JPM(JK).GT.NCOLMN)  BASE(J2,JK) = 1.0
      IF(JPM(JK).LE.NCOLMN)  GOTO 310
      J2 = J2 + 1
      GOTO 300
310   DO 311 J=1,MROWS1
      J1 = J1 + 1
311   BASE(J,JK) = OUT2(J1)
300   CONTINUE
C
C    THE FOLLOWING IS FOR OUTPUT.
C
      WRITE(6,109)
      WRITE(6,101) ((BASE(I,J),J=1,MROWS1),I=1,MROWS1)
      WRITE(2,101) ((BASE(I,J),J=1,MROWS1),I=1,MROWS1)
C
C  IF YOU WANT THE BASIS INVERSE PUNCHED OUT
C     (THERE ARE THREE (3) CARDS FOR THE BASIS INVERSE)
C     REMOVE THE 'C' FROM THE FOLLOWING CARD.
C     WRITE(7,101) ((BASE(I,J),J=1,MROWS1),I=1,MROWS1)
C
202   CONTINUE
      WRITE(6,107)
      WRITE(6,108) (JPM(JK), JK=1,NTOT)
      WRITE(2,108) (JPM(JK), JK=1,NTOT)
C
```

```
C  IF YOU WANT THE ORDER OF THE ITH BASIC VARIABLE, JPM(I),
   PUNCHED OUT
C     REMOVE THE 'C' FROM THE FOLLOWING CARD.
C     WRITE(7,108) (JPM(JK), JK=1,NTOT)
C
      WRITE(2,101) (OUT(J),J=1,NCOLMN)
C
C  IF YOU WANT THE OPTIMAL INTEGER SOLUTION PUNCHED OUT
C     REMOVE THE 'C' FROM THE FOLLOWING CARD.
C     WRITE(7,101) (OUT(J),J=1,NCOLMN)
C
C
C  IF YOU WANT TO SUPPLY YOUR OWN FEASIBLE INTEGER SOLUTION
C     REMOVE THE 'C' FROM THE FOLLOWING CARD.
C     GOTO 151
C
C
C  TO OBTAIN THE STARTING INTEGER SOLUTION TO THE ILP
C
      DO 110 I=1,MROWS1
      READ( 3,111)  (A(I,J),J=1,NCOLMN)
110   CONTINUE
      READ( 3,111) (B(I),I=1,MROWS1)
      READ(3,111)(CC(J),J=1,NCOLMN)
111   FORMAT(15F5.0)
      DO 1000 I=1,NCOLMN
      XF(I) = SNGL(OUT(I))
      IF(CC(J)) 1001,1000,1000
1001  IF (XF(I).EQ.AINT(XF(I))) GOTO 1000
      XF(I) = XF(I) + 1.
1000  CONTINUE
      DO 1020 I=1,NCOLMN
1020  XF(I) = AINT(XF(I))
      ITEST = 0
C
C  SET THE MAXIMUM NUMBER OF ITERATIONS.
C
1021  IF(10 -ITEST) 1100,1022,1022
1022  ITEST = ITEST + 1
C
C  CHECK TO SEE IF THE POINT IS FEASIBLE.
C
      DO 1030 I=1,MROWS1
      SUM = 0.0
      DO 1025 J=1,NCOLMN
      SUM = SUM + A(I,J) * XF(J)
1025  CONTINUE
      IF(B(I)-SUM) 1026,1030,1030
1026  ITER = I
C
C  IF POINT IS NOT FEASIBLE, WRITE THE CONSTRAINT VIOLATED AND
C     THE POINT THAT VIOLATED THE CONSTRAINT.
```

```
C
      WRITE(6,1027) I,B(I),SUM,(XF(IJ),IJ=1,NCOLMN)
1027  FORMAT( 10X,10H          B(I2,21H)                    SUM   /
     1        15X, 2F12.2 // 14H THE POINT IS    /4(15F8.2/)//////)
      GOTO 1035
1030  CONTINUE
      GOTO 1120
1035  L  = 0
      N1 = 0
C
C  FIND ALL OF THE NONZERO VALUES OF THE VARIABLES.
C     IDX1 STORES ALL OF THE ZERO VALUED VARIABLES.
C     IDXF STORES ALL OF THE NONZERO VALUED VARIABLES.
C
      DO 1040 J=1,NCOLMN
      IF( XF(J) ) 1036,1037,1038
1036  XF(J) = 0.0
1037  L = L+1
      IDX1(L) = J
      GOTO 1040
1038  N1 = N1 + 1
      XFMIN(N1) = XF(J)
      IDXF(N1) = J
1040  CONTINUE
      IF(L.LT.1)  GOTO 1042
C
C  TACK ON TO THE END OF THE INDES OF THE NONZERO VARIABLES THE
C     INDEX OF THE ZERO VARIABLES.
C
      DO 1041 I=1,L
1041  IDXF(N1+I) = IDX1(I)
C
C  IF THERE IS AT LEAST ONE NONZERO VARIBALE,
C     THEN ARRANGE THE NONZERO VARIABLES FROM
C     MINIMUM TO MAXIMUM VALUE.
C     IDX2 IS THE INDEX OF THE REARRANGED VARIABLES.
C
1042  IF(N1.GT.0) GOTO 1045
      DO 1044  I=1,L
1044  XFMIN(I)=0.0
      GOTO 1046
1045  CALL SORTI(XFMIN,IDX2,1,N1)
1046  IF(N1.EQ.NCOLMN)GOTO 1048
      N1P1 = N1 + 1
      DO 1047 I=N1P1,NCOLMN
1047  IDX2(I) = IDXF(I)
C
C  ROUNDING PROCEDURE.
C     TAKE THE SMALLEST NONZERO VARIABLE,
C       IF THE CORRESPONDING CONSTRAINT COEFFICIENTS IS
C          POSITIVE -- ROUND DOWN
C          NEGATIVE -- ROUND UP.
C
```

```
1048  DO 1060 KL=1,NCOLMN
      IV = IDXF(IDX2(KL))
      LCK = 0
1049  IF( A(ITER,IV) ) 1050,1060,1051
1050  XF(IV) = XF(IV) + 1.0
      GOTO 1055
1051  XF(IV) = XF(IV) - 1.0
      IF(XF(IV).LT.0.0) XF(IV) = 0.0
1055  SUM = 0.0
      DO 1056 I=1,NCOLMN
      SUM = SUM + A(ITER,I) * XF(I)
1056  CONTINUE
C
C
C     IF THIS DOES NOT STAISFY THE CONSTRAINT,
C       THEN TAKE THE NEXT SMALLEST VARIABLE
C       AND REPEAT THE PROCEDURE.
C
C  IF THE CONSTRAINT IS SATISFIED, THEN USE THIS POINT TO
C     CHECK ALL OF THE OTHER CONSTRAINTS.
C
C
      IF( B(ITER) - SUM) 1057,1021,1021
1057  LCK = LCK + 1
C
C  SET THE MAXIMUM NUMBER OF CHANGES FOR ONE VARIABLE FOR ONE
   ITERATION.
C     CHANGE THIS IF THERE IS AN OSCILATION BACK AND FORTH BETWEEN
C     TWO POINTS.
C
      IF(LCK.LT.1) GOTO 1049
1060  CONTINUE
      GOTO 1021
1100  CONTINUE
      WRITE(6,1105) ITEST
1105  FORMAT(47H NO FEASIBLE STARTING INTEGER SOLUTION HAS BEEN
     1       26H FOUND AT THIS POINT AFTER   I5,6H TRYS.)
      GOTO 1150
1120  CONTINUE
      OBJ = 0.0
      DO 1122 I=1,NCOLMN
1122  OBJ = OBJ + XF(I)*CC(I)
      WRITE(6,1125) (XF(I),I=1,NCOLMN)
1125  FORMAT(53H A FEASIBLE STARTING INTEGER SOLUTION HAS BEEN
     FOUND. /
     14(15F8.2/))
      WRITE(6,1126) OBJ
1126  FORMAT(    40H THE VALUE OF THE OBJECTIVE FUNCTION IS
     / F12.4 )
      WRITE(2,101)  (XF(J),J=1,NCOLMN)
C
C  IF YOU WANT THE OPTIMAL 'FEASIBLE' INTEGER SOLUTION PUNCHED OUT
```

```
C      REMOVE THE 'C' FROM THE FOLLOWING CARD.
C      WRITE(7,101)  (XF(J),J=1,NCOLMN)
C
       GOTO 151
100    WRITE(6,106)
106    FORMAT( ' AN ERROR HAS OCCURED')
107    FORMAT(' JPM(I) IN THE ORDER THEY SHOULD OCCUR AND IS THE
     INDEX OF 1 THE ITH BASIC VARIABLE')
108    FORMAT(15I4)
109    FORMAT(////' THE BASIS INVERSE IS '//)
1150    CONTINUE
       I = 10
       WRITE(10,1151) I
1151    FORMAT(I5)
151    CONTINUE
       I = 0
       WRITE(10,1151) I
200    RETURN
       END

       SUBROUTINE SORTI(A,ID,II,JJ)
       DIMENSION A(1),IU(30),IL(30),ID(1)
       INTEGER T1,T2
       M=1
       I=II
       J=JJ
       DO 1 IS=I,J
1      ID(IS)=IS
5      IF(I .GE. J) GOTO 70
10     K=I
       IJ=(J+I)/2
       T=A(IJ)
       T1=ID(IJ)
       IF(A(I) .LE. T) GOTO 20
       A(IJ)=A(I)
       ID(IJ)=ID(I)
       A(I)=T
       ID(I)=T1
       T=A(IJ)
       T1=ID(IJ)
20     L=J
       IF(A(J) .GE. T) GOTO 40
       A(IJ)=A(J)
       ID(IJ)=ID(J)
       A(J) =T
       ID(J)=T1
       T=A(IJ)
       T1=ID(IJ)
       IF(A(I) .LE. T) GOTO 40
       A(IJ)=A(I)
       ID(IJ)=ID(I)
       A(I)=T
       ID(I)=T1
```

```
      T=A(IJ)
      T1=ID(IJ)
      GOTO 40
30    A(L)=A(K)
      ID(L)=ID(K)
      A(K)=TT
      ID(K)=T2
40    L=L-1
      IF(A(L) .GT. T) GOTO 40
      TT=A(L)
      T2=ID(L)
50    K=K+1
      IF(A(K) .LT. T) GOTO 50
      IF(K .LE. L) GOTO 30
      IF(L-I .LE. J-K) GOTO 60
      IL(M)=I
      IU(M)=L
      I=K
      M=M+1
      GOTO 80
60    IL(M)=K
      IU(M)=J
      J=L
      M=M+1
      GOTO 80
70    M=M-1
      IF(M .EQ. 0) RETURN
      I=IL(M)
      J=IU(M)
80    IF(J-I .GE. 11) GOTO 10
      IF(I .EQ. II) GOTO 5
      I=I-1
90    I=I+1
      IF(I .EQ. J) GOTO 70
      T=A(I+1)
      T1=ID(I+1)
      IF(A(I) . LE. T) GOTO 90
      K=I
      ID(K)=ID(I)
100   A(K+1)=A(K)
      ID(K+1)=ID(K)
      K=K-1
      IF(T .LT. A(K)) GOTO 100
      A(K+1)=T
      ID(K+1)=T1
      GOTO 90
      END


//LKED.SYSLIB DD DSN=LLOAD,DISP=(SHR,KEEP)
//            DD DSN=SYS1.FORTLIB,DISP=SHR
//LKED.SYSLMOD DD DSN=&MCCAL,DISP=(NEW,PASS),UNIT=SYSDA,
```

```
//    SPACE=(CYL,(1,1,10)),DCB=(DSORG=PO,RECFM=U,BLKSIZE=3625)
//LKED.SYSIN DD *
     INSERT READCOMM
     ENTRY MAIN
     NAME DATAHILL(R)



//CPC EXEC PGM=COMPILER
//STEPLIB DD DISP=(SHR,KEEP),DSNAME=LLOAD
//SCRATCH1 DD UNIT=SYSDA,DISP=(NEW,DELETE),SPACE=(TRK,(1,1))
//SCRATCH2 DD UNIT=SYSDA,DISP=(NEW,DELETE),SPACE=(TRK,(1,1))
//SCRATCH3 DD UNIT=SYSDA,DISP=(NEW,DELETE),SPACE=(TRK,(1,1))
//SCRATCH4 DD UNIT=SYSDA,DISP=(NEW,DELETE),SPACE=(TRK,(1,1))
//SYSMLCP DD UNIT=SYSDA,SPACE=(TRK,(2,1)),DISP=(NEW,PASS)
//SYSPRINT DD  SYSOUT=A
//SYSPUNCH DD SYSOUT B
//SYSIN    DD  *
         PROGRAM ('ND')
         INITIALZ
         MOVE(XDATA,'OPTBAS')
         MOVE(XPBNAME,'MYFILE')
         MOVE(XOBJ,'O')
         MOVE(XRHS,'CONST')
         ASSIGN ( 'COMMFMT','FT04F001','COMM')
         PREPOUT( 'COMMFMT' )
         CONVERT('SUMMARY')
         BCDOUT
         SETUP(1)
         PRIMAL
         SOLUTION
         TRANCOL('ENTIRE','INVERSE')
         TRANCOL('ENTIRE')
         TRANCOL ('FILE','COMMFMT','ENTIRE')
         SOLUTION('FILE','COMMFMT')
         DATAHILL
         EXIT
         PEND



//EXEC   EXEC  PGM=EXECUTOR,COND=(0,NE,CPC),REGION=220K,TIME=3
//STEPLIB DD DISP=(SHR,KEEP),DSNAME=LLOAD
//         DD DSN=&MCCAL,DISP=(OLD,PASS),UNIT=SYSDA
//SCRATCH1 DD UNIT=SYSDA,DISP=(NEW,DELETE),SPACE=(CYL,(1,1))
//SCRATCH2 DD UNIT=SYSDA,DISP=(NEW,DELETE),SPACE=(CYL,(1,1))
//PROBFILE DD UNIT=SYSDA,SPACE=(CYL,(1,1)),DISP=(NEW,DELETE)
//ETA1     DD UNIT=SYSDA,SPACE=(CYL,(1,1)),DISP=(NEW,DELETE)
//MATRIX1  DD UNIT=SYSDA,SPACE=(CYL,(1,1)),DISP=(NEW,DELETE)
```

```
//SYSMLCP DD UNIT=SYSDA,DSNAME=*.CPC.SYSMLCP,DISP=(OLD,DELETE)
//SYSPRINT DD  SYSOUT=A
//EXEC.FT02F001 DD DSN=*.S1.GO.FT02F001,DISP=(MOD,PASS)
//EXEC.FT03F001 DD DSN=*.S1.GO.FT03F001,DISP=(OLD,DELETE)
//EXEC.FT06F001 DD SYSOUT=A
//EXEC.FT07F001 DD SYSOUT=B
//EXEC.FT04F001 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPUNCH DD SYSOUT=B
//EXEC.FT10F001 DD UNIT=SYSDA,SPACE=(TRK,(1,1),RLSE),
// DISP=(,PASS),
// DCB=(RECFM=FBS,LRECL=80,BLKSIZE=880)
//SYSIN DD DSN=*.S1.GO.FT01F001,DISP=(OLD,DELETE)
//CHKF EXEC FORTGCLG,REGION.GO=42K,TIME=1
//FORT.SYSLIN DD DISP=(NEW,PASS)
//FORT.SYSIN DD  *
      READ(10,100) I
100   FORMAT(I5)
      IF(I) 20,20,10
10    STOP 10
20    CONTINUE
      STOP
      END
//GO.FT10F001 DD DSN=*.EXEC.FT10F001,DISP=(OLD,DELETE)



//TWO EXEC PGM=OPTALG,COND=(2,LT,CHKF.GO),REGION=(,336K),TIME=2
//STEPLIB DD DSN=LLOAD,DISP=SHR
//FT06F001 DD SYSOUT=A
//FT05F001  DD DSN=*.S1.GO.FT02F001,DISP=(OLD,DELETE)



//SFAIL EXEC FORTGCLG,COND.FORT=(0,EQ,CHKF.GO),
// COND.LKED=((0,EQ,CHKF.GO),(4,LT,FORT)),
// COND.GO=((0,EQ,CHKF.GO),(4,LT,FORT),(4,LT,LKED)),
// REGION.GO=42K,TIME=1
//FORT.SYSLIN DD DISP=(NEW,PASS)
//FORT.SYSIN DD *
      REAL X(80)
5     READ(5,10,END=100) X
10    FORMAT(80A1)
      WRITE(7,10) X
      GOTO 5
100   STOP
      END
//GO.FT05F001 DD DSN=*.S1.GO.FT02F001,DISP=(OLD,DELETE)
```

NELC ZANGWL

CATALOG IDENTIFICATION:

E4 NELC ZANGWL

PROGRAMMERS:

D. C. McCALL, Decision and Control Technology Division, and
C. M. BECKER, Applications Software Division

PURPOSE:

To compute the minimum of a function $f(x_1, \ldots, x_n)$, of n real variables

RESTRICTIONS AND LIMITATIONS:

A maximum of 20 variables can be handled.

LANGUAGE:

FORTRAN IV

COMPUTER CONFIGURATION:

IBM 360/65
Core storage: 19086 bytes

ENTRY POINTS:

ZANGWL

SUBPROGRAMS AND WHERE REFERENCED:

User-supplied programs
FUNC called by ZANGWL, (POWELL)
Programmer-supplied programs
ZGITER called by ZANGWL
POWELL called by ZANGWL

USAGE:

CALL ZANGWL (XI, N, EACCUR, QSTEP, ISTOP, LPRINT, IX, LPUNCH, XOPT, FF)

For a description of parameters see the listing.

INPUT FORMAT:

All input is through the parameter list except when user-supplied search directions are desired. Then ZANGWL expects N vectors of length N input on cards in 4(F15.10, 5X) Format, where N = n is the number of variables.

OUTPUT:

The output depends on the print and punch option. See the listing.

ERROR MESSAGES:

None

PROGRAM DESCRIPTION:

ZANGWL – acts as a driver and convergence monitor for ZGITER. If the vector $x = (x_1, x_2, \ldots, x_n)$ on returning from ZGITER is to within EACCUR of the value on entering, ZANGWL returns.

ZGITER – keeps track of the directions to be searched and normalizes each newly generated direction.

POWELL – finds the minimum of the objective function along a direction supplied by ZGITER using quadratic interpolation.

MATHEMATICAI METHOD:

ZANGWL is based on a method proposed by W. I. Zangwill in the Computer Journal, Vol. 10, 1967, pp. 293-296.[13] The method is outlined as follows: Let $f(x_1, x_2, \ldots, x_n)$ be the function to be minimized, and c_r, $r = 1, \ldots, n$ be the unit coordinate directions. Assume that an initial point p_n^o and n normalized directions ξ_r^1 $r = 1,2, \ldots, n$ are given.

To initialize, calculate λ_n^o to minimize $f(p_n^o + \lambda_n^o \xi_n^1)$, then set $p_{n+1}^1 = p_n^o + \lambda_n^o \xi_n^1$, $t = 1$ and go to iteration k with $k = 1$.

Iteration k: p_{n+1}^{k+1}, ξ_r^k, $r = 1, \ldots, n$ and t are given.

Step (i): Find α to minimize $f(p_{n+1}^{k-1} + \alpha c_i)$. Update t by

$$t = \begin{cases} t + 1 \text{ if } 1 \leqslant t < n \\ \\ i \quad \text{if} \quad t = n \end{cases}$$

If $\alpha \neq 0$, let $p_\jmath^k = p_{n+1}^{k-1} + \alpha c_t$. If $\alpha = 0$, repeat step (i). Should step (i) be repeated n times in succession, stop; the point p_{n+1}^{k-1} is optimal.

Step (ii): For $r = 1, \ldots, n$ calculate λ_r^k to minimize $f(p_{r-1}^k + \lambda_r^k \xi_r^k)$ and define $p_r^k = p_{r-1}^k + \lambda_r^k \xi_r^k$. Let $\xi_{n+1}^k = (p_n^k - p_{n+1}^{k-1})/\|p_n^k - p_{n+1}^{k-1}\|$

Determine λ_{n+1}^k to minimize $f(p_n^k + \lambda_{n+1}^k \xi_{n+1}^{k-1})$ and set $p_{n+1}^k = p_n^k + \lambda_{n+1}^k \xi_{n+1}^k$.

Define $\xi_r^{k+1} = \xi_{r+1}^k$, $r = 1, \ldots, n$

Go to iteration k with k+1 replacing k.[13]

NUMERICAL EXAMPLES – GENERAL TEST FUNCTIONS

In this section we list the functions for which ZANGWL computed the minimum, and summarize the results. These test functions are selected because the surfaces they define have steep curving valleys or have many known optima. The results are tabulated for each function. This table lists the various initial points, the computed optimum, the minimum value of the objective function, the number of function evaluations, and the elapsed CPU in seconds' time on the IBM 360/65.

In comparison with minimization routines (FP and CNJGAT) at NELC,[12] the computing times are a bit slower and many function evaluations are needed, but with the speed of the 360/65 this is relatively insignificant compared with the number of man-hours spent in deriving the analytic expression for the gradient.

1. Rosenbrock's[60] function: $f(x_1,x_2) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2$. This function has a steep valley along the parabola $x_2 = x_1^2$ with a minimum at (1,1).

2. Cube:[15] $f(x_1,x_2) = 100(x_2 - x_1^3)^2 + (x_1 - 1)^2$. Cube is similar to Rosenbrock's function except the steep valley follows the curve $x_2 = x_1^3$ and has a minimum (1,1).

3. Helical:[9] $f(x_1,x_2,x_3) = 100\,(x_3 - 100)^2 + (r - 1)^2 + x_3^2$ where $x_1 = r \cos 2\pi\theta$, $x_2 = r \sin 2\pi\theta$, and $r = \sqrt{x_1^2 + x_2^2}$. This function has a steep helical valley with a minimum at (1,0,0).

4. THREE:[61] $f(x_1,x_2,x_3) = \dfrac{-1}{1 + (x_1 - x_2)^2} - \sin(\tfrac{1}{2}\pi x_2 x_3)$

$$-\exp\left(-\left(\frac{x_1 + x_2}{x_2}\right)-2\right)^2$$

. THREE has minima at $x_1 = x_2 = x_3 = \pm\sqrt{4n+1}$, $n \geq 0$ integral with a minimum value of -3. This function tends to change quickly from the point (0,1,2) and then flattens out until it reaches an optimum. The optimum depends on the starting point x_0.

5. FOUR:[62] $f(x_1,x_2,x_3,x_4) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$. FOUR has its minimum at (0,0,0,0).

6. CHEBYQ(UAD):[63] This relatively new function allows testing a routine on a function with an arbitrary number of variables; i.e., CHEBYQ

$(x_1, \ldots, x_n)$, where n is a parameter preset by the user. For n = 1,2, . . . , 7,9 the minimum value of CHEBYQ is zero; however, for other n the minimization is still valid.

Briefly, CHEBYQ does the following: Let $x = (x_1, \ldots, x_n)$ be a vector (abscissae) whose coordinates are in the range $0 \leq x_i \leq 1$. Then, choosing the shifted chebyshev polynomial T_j, we define:

$$\Delta_i(\underline{x}) = \int_0^1 T_i(z)dz - \frac{1}{n}\sum_{j=1}^{n} T_i(x_j)$$

Then the function $f(x) = \sum_{i=1}^{n} (\Delta_i(\underline{x}))^2$ has the property that if X is the vector of abscissae, then f = 0; otherwise, f > 0. Although contrived, CHEBYQ is a good example of a complicated objective function that can occur. The FORTRAN IV listing of CHEBYQ follows.

```
      SUBROUTINE CHEBYQ(F,X,N)
      IMPLICIT REAL*8(A-H,O-Z)
      LOGICAL IEVEN
      DIMENSION Y(20),TI(20),TIMIN(20),X(1)
      DELTA = 0.0000
      ZERO= 0.0000
      ONE = 1.0000
      TWO = 2.0000
      DO 10 J=1,N
      Y(J) = TWO*X(J)-ONE
      DELTA= DELTA  +Y(J)
      TI(J) = Y(J)
10    TIMIN(J) = ONE
      F      = DELTA * DELTA
      IEVEN = .FALSE.
      DO 20 I=2,N
      IEVEN = .NOT.IEVEN
      DELTA = ZERO
      DO 19 J=1,N
      TIPLUS = TWO*Y(J)*TI(J)-TIMIN(J)
      DELTA  = DELTA + TIPLUS
      TIMIN(J)= TI(J)
19    TI(J)  = TIPLUS
      A = ZERO
      IF(IEVEN)  A= -ONE/(I*I-ONE)
      DELTA = DELTA/N-A
20    F = F+ DELTA*DELTA
      RETURN
      END
```

ZANGWL - ROSIE

optimum at (1.0, 1.0)

QSTEP = 0.1; EACCUR = 1.0D-04

Initial Point	Computed Optimum	Objective Function	Number of Evaluations	Time (sec)
1 -1.2 2 1.0	0.10000000D01 0.10000000D01	0.16961904D-22	325	0.26
1 -1.2 2 -1.0	0.10000000D01 0.10000000D01	0.16961905D-22	328	0.20
1 1.2 2 -1.0	0.10000000D01 0.10000000D01	0.84351733D-24	206	0.11

QSTEP = 0.01; EACCUR = 1.0D-04

1 -1.2 2 1.0	0.99999991D00 0.99999992D00	0.77935877D-14	366	0.19
1 -1.2 2 -1.0	0.99999991D00 0.99999992D00	0.77935878D-14	370	0.19
1 1.2 2 -1.0	0.10000000D01 0.10000000D01	0.23682822D-14	152	0.12

ZANGWL - CUBE

optimum at (1.0, 1.0)

QSTEP = 0.1; EACCUR = 1.0D-04

Initial Point	Computed Optimum	Objective Function	Number of Evaluations	Time (sec)
1 -1.2 2 1.0	0.99999999D00 0.99999999D00	0.31134541D-21	402	0.28
1 -1.2 2 -1.0	0.99999999D00 0.99999999D00	0.31134835D-21	399	0.23
1 1.2 2 -1.0	0.99999999D00 0.99999999D00	0.78369023D-21	179	0.17

QSTEP = 0.01; EACCUR = 1.0D-04

1 -1.2 2 1.0	0.10000000D01 0.10000000D01	0.37920987D-25	473	0.25
1 -1.2 2 -1.0	0.10000000D01 0.10000000D01	0.37920987D-25	470	0.25
1 1.2 2 -1.0	0.10000000D01 0.10000000D01	0.27144901D-25	216	0.12

ZANGWL - THREE

optima at $x_1 = x_2 = x_3 = \pm\sqrt{4n+1}$, n integral

QSTEP = 0.1; EACCUR = 1.0D-04

Initial Point	Computed Optimum	Objective Function	Number of Evaluations	Time (sec)
1 0.0 2 1.0 3 2.0	0.10000000D01 0.99999999D00 0.99999999D00	-0.30000000D01	288	0.38
1 0.0 2 -1.0 3 -2.0	-0.10000000D01 -0.10000000D01 -0.99999999D00	-0.30000000D01	298	0.30
1 0.0 2 1.0 3 -2.0	0.40012498D02 0.40012498D02 0.40012498D02	-0.30000000D01	5710	4.88

STEP = 0.01; EACCUR = 1.0D-04

1 0.0 2 1.0 3 2.0	0.10000000D01 0.10000000D01 0.99999999D00	-0.30000000D01	257	0.28
1 0.0 2 -1.0 3 -2.0	-0.10000000D01 -0.10000000D01 -0.99999999D00	-0.30000000D01	257	0.35

ZANGWL - HELICAL

optimum at (1.0, 0.0, 0.0)

QSTEP = 0.1; EACCUR = 1.0D-04

Initial Point	Computed Optimum	Objective Function	Number of Evaluations	Time (sec)
1 -1.0 2 0.0 3 0.0	0.10000000D01 -0.34372772D-15 -0.81379053D-15	0.29631116D-30	462	0.56
1 1.0 2 1.0 3 1.0	0.10000000D01 -0.55742421D-16 -0.90526145D-16	0.77927466D-32	366	0.38
1 1.5 2 0.5 3 -0.5	0.10000000D01 0.86066593D-17 0.16356615D-16	0.18577895D-33	351	0.38

QSTEP = 0.01; EACCUR = 1.0D-04

1 -1.0 2 0.0 3 0.0	0.10000000D01 -0.54445252D-21 0.11963532D-16	0.14457809D-31	622	0.59
1 1.0 2 1.0 3 1.0	0.10000000D01 -0.50124625D-17 -0.77793682D-17	0.60518735D-34	375	0.42
1 1.5 2 0.5 3 -0.5	0.10000000D01 0.37411888D-20 -0.31920217D-18	0.10674562D-34	337	0.33

ZANGWL - FOUR

optimum at (0.0, 0.0)

QSTEP = 0.1 ; EACCUR = 1.0D-04

Initial Point	Computed Optimum	Objective Function	Number of Evaluations	Time (sec)
1 3.0	-0.13754371D-5	0.68469142D-23	1350	0.95
2 -1.0	0.13754371D-6			
3 0.0	-0.57715785D-6			
4 1.0	-0.57715786D-6			
1 -3.0	-0.71682591D-6	0.36195490D-18	1169	0.73
2 -1.0	0.71682590D-7			
3 2.0	0.10589914D-4			
4 1.0	0.10589913D-4			
1 3.0	0.58159117D-5	0.79920637D-20	1568	0.97
2 1.0	-0.58159117D-6			
3 -2.0	0.44312920D-5			
4 -1.0	0.44312921D-5			

QSTEP = 0.01 ; EACCUR = 1.0D-04

1 3.0	0.46374739D-6	0.35002584D-20	781	0.57
2 -1.0	-0.46374741D-7			
3 0.0	-0.32170026D-5			
4 1.0	-0.32170025D-5			
1 -3.0	-0.40710970D-7	0.79074373D 24	1503	0.95
2 -1.0	0.40710966D-8			
3 2.0	0.40141593D-6			
4 1.0	0.40141591D-6			
1 3.0	0.29354616D-6	0.36784227D-23	1224	0.76
2 1.0	-0.29354616D-7			
3 -2.0	-0.45198631D-6			
4 -1.0	-0.45198633D-6			

ZANGWL - CHEBYQUAD

QSTEP = 0.01; EACCUR = 1.0D-04

Initial Point	Computed Optimum	Objective Function	Number of Evaluations	Time (sec)
		n = 6		
1 0.142	0.66876591D-1			
2 0.285	0.28874067D00			
3 0.428	0.36668229D00	0.32039347D-17	704	1.81
4 0.571	0.63331770D00			
5 0.714	0.71125932D00			
6 0.857	0.93312341D00			
		n = 8		
1 0.111	0.43152760D-1			
2 0.222	0.19309084D00			
3 0.333	0.26632870D00			
4 0.444	0.50000000D00	0.35168737D-02	2007	7.99
5 0.555	0.49999999D00			
6 0.666	0.73367129D00			
7 0.777	0.80690916D00			
8 0.888	0.95684724D00			
		n = 10		
1 0.090	0.59619901D-1			
2 0.181	0.16670828D00			
3 0.272	0.23917065D00			
4 0.363	0.39888429D00			
5 0.454	0.39888429D00	0.65039548D-02	2603	14.30
6 0.545	0.60111570D00			
7 0.636	0.60111570D00			
8 0.727	0.76082934D00			
9 0.818	0.83329171D00			
10 0.909	0.94038009D00			

```
C       SUBROUTINE ZANGWL
C
C       PURPOSE
C          TO FIND THE MINIMUM OF A REAL VALUED FUNCTION OF N-VARIABLES
C          WHOSE VALUES ARE UNCONSTRAINED.
C
C       USAGE
C          CALL ZANGWL (XI,N,EACCUR,QSTEP,ISTOP,LPRINT,IX,LPUNCH,XOPT,FF)
C
C       DESCRIPTION OF PARAMETERS
C          XI     - THE INITIAL GUESS OF THE OPTIMUM
C          N      - THE NUMBER OF VARIABLES
C          EACCUR- THE INITIAL ACCURACY DESIRED.  FOR BEST RESULTS SET
C                   LESS THAN QSTEP**5.
C          QSTEP  - THE INITIAL STEP SIZE FOR THE ONE DIMENSIONAL SEARCH
C                   ROUTINE POWELL. QSTEP EQUAL .1 WORKS WELL.
C          ISTOP  - QSTEP REDUCTION CODE
C                   0 NO REDUCTION
C                   1 AFTER FINDING AN MINIMUM TO WITHIN EACCUR, QSTEP IS
C                   SET TO EACCUR,EACCUR=EACCUR**2 AND THE ROUTINE DOES ONE
C                   FINAL MINIMIZATION TO WITHIN EACCUR.
C          LPRINT- PRINT CODE
C                   0 DO NOTHING
C                   1 EACH N-DIMENSIONAL ITERATION ONLY
C                   2 FINAL RESULTS ONLY
C                   3 BOTH 1+2
C                   4 3 PLUS THE POINT ON ENTERING AND LEAVING POWELL.
C          IX     - CHOICE OF USER SUPPLIED DIRECTIONS. IF.EQ.0,CO-CRDINATE
C                   DIRECTIONS ARE USED.
C          LPUNCH- A PUNCH OPTION, IF DIFFERENT FROM 0, ZANGWL WILL PUNCH
C                   OUTPUT FOR THE MINIMUM POINT.
C          XOPT   - A LIST OF LENGTH N CONTAINING THE MINIMUM.
C          FF     - FF = FUNC (XOPT,N)
C
C          NOTE ALL PARAMETERS ARE DOUBLE PRECISION.
C
C       SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED
C          USER MUST SUPPLY THE FUNCTION SUBPROGRAM FUNC(X,N)
C
C       METHOD
C          THIS ROUTINE IS BASED ON A METHOD PROPOSED BY W.I.ZANGWILL,
C          COMPUTER JOUR.,VOL.10, 1967, P293-296.
C
C       ..................................................................
C
        SUBROUTINE ZANGWL(NEW,N,EACCUR,QSTEP,ISTOP,LPRINT,IX,LPUNCH,XOPT,
      1 FF)
        IMPLICIT REAL*8 (A-H,O-Z)
        DIMENSION NEW(1), OLD(20), XI(20,21), D(20), PT(20), XINEW(20)
        DIMENSION R(20), XJ(20,21), XOPT(1)
        INTEGER T, OBJFN, COUNT
        REAL*8 LAMBDA,NEW,MINFN
        COMMON/ZANG/EPSILN,Q,E,IPOWEL,OBJFN,LIST,COUNT
```

```
      IP=0
      Q = QSTEP
      E = EACCUR
      LIST = LPRINT
      IBMCRD= LPUNCH
      OBJFN=0
      IPOWEL=2
      EPSILN=1.0D-15
C     IP=STOP FLAG TERMINATING SIMULTANEOUS REDUCTION OF BOTH Q AND E
C     K=NUMBER OF CURRENT ITERATION
C     OBJFN=CURRENT QUANTITY OF OBJECTIVE FUNCTION EVALUATIONS
C     OLD(IC)=IC-TH COMPONENT OF ENTER POINT FOR ZGITER
C     NEW(IC)=IC-TH COMPONENT OF INITIAL POINT AND POINT COMPUTED IN
C        ZGITER
C     R(IC)=OLD(IC) AT START OF ZGITER
C     XI(IC,ID)=IC-TH COMPONENT OF ID-TH NORMALIZED NONCOORDINATE
C        DIRECTION
C     XJ(IC,ID)=XI(IC,ID) AT START OF ZGITER
C     XINEW(IC)=IC-TH COMPONENT OF (N+1)-ST ('EXTRA')
C        NORMALIZED NONCOORDINATE DIRECTION
C     PT(IC)=IC-TH COMPONENT OF MINIMUM POINT OF N-DIMENSIONAL
C        MINIMIZATION
C     D(IC)=IC-TH COMPONENT OF NORMALIZED NONCOORDINATE DIRECTION OF
C        ONE DIMENSIONAL MINIMIZATION
C     ALPHA=MINIMUM STEP LENGTH FOR N-DIMENSIONAL MINIMIZATION
C     LAMBDA=MINIMUM STEP LENGTH FOR ONE DIMENSIONAL MINIMIZATION
C     MINFN=OBJECTIVE FUNCTION MINIMUM VALUE
C     ZGFN=CURRENT ZGITER ITERATION OBJECTIVE FUNCTION VALUE
C     STFN=OBJECTIVE FUNCTION VALUE AT INITIAL POINT
C     JUMP=FIRST ZGITER ITERATION FLAG
C     OBJ=OBJFN AT START OF ZGITER
C     PTDIF=MAXIMUM NEW AND OLD POINT COMPONENT DIFFERENCE
C
10    CONTINUE
      IF (IP.GT.0) GO TO 20
20    IF (IP.GT.0) GO TO 30
      STFN=FUNC(NEW,N)
      FF=STFN
      OBJFN=OBJFN+1
      GO TO 40
30    STFN=FUNC(PT,N)
      OBJFN=OBJFN+1
40    IF (LIST.EQ.0) GO TO 90
      PRINT 390, N,Q,E
      PRINT 480, LIST,IBMCRD,IX
      IF (IP.EQ.0) GO TO 50
      PRINT 400, (IC,PT(IC),IC=1,N)
      GO TO 60
50    PRINT 400, (IC,NEW(IC),IC=1,N)
60    PRINT 530, STFN
      IF (IP.EQ.0) GO TO 70
      GO TO 80
70    CONTINUE
```

```
80      IF (IP.GT.0) GO TO 130
        IF (IX.EQ.0) GO TO 90
C
C       USER SUPPLIES INITIAL NONCOORDINATE DIRECTIONS
C
        READ 340, ((XI(IC,ID),IC=1,N),ID=1,N)
        GO TO 130
C
C       COMPUTE INITIAL NONCOORDINATE DIRECTIONS
C
90      DO 120 ID=1,N
        DO 110 IC=1,N
        IF (IC.EQ.ID) GO TO 100
        XI(IC,ID)=0.0D00

        GO TO 110
100     XI(IC,ID)=1.0D00
110     CONTINUE
120     CONTINUE
C
C       MINIMIZE IN ONE DIMENSION USING INITIALIZATION DATA
C
130     DO 140 IC=1,N
140     D(IC)=XI(IC,N)
        IF (IP.EQ.0) GO TO 160
        IF (LIST.EQ.0) GO TO 145
        PRINT 580, (IC,PT(IC),IC=1,N)
145     CALL POWELL (N,OLD,D,LAMBDA,FF)
        DO 150 IC=1,N
150     OLD(IC)=OLD(IC)+LAMBDA*D(IC)
        IF (LIST.EQ.0) GO TO 155
        PRINT 590, (IC,OLD(IC),IC=1,N)
155     CONTINUE
        GO TO 180
160     IF (LIST.EQ.0) GO TO 165
        PRINT 560, (IC,NEW(IC),IC=1,N)
165     CALL POWELL (N,NEW,D,LAMBDA,FF)
C
C       COMPUTE FIRST POINT
C
        DO 170 IC=1,N
170     OLD(IC)=NEW(IC)+LAMBDA*D(IC)
        IF (LIST.EQ.0) GO TO 175
        PRINT 600, (IC,OLD(IC),IC=1,N)
175     CONTINUE
C
C       INITIALIZE FOR FIRST ITERATION
C
        K=1
        T=1
        JUMP=1
180     IF (IX.EQ.0) GO TO 190
        JUMP=JUMP+1
```

```
190     DO 200 IJ=1,N
200     R(IJ)=OLD(IJ)
        DO 220 IK=1,N
        DO 210 IL=1,N
210     XJ(IL,IK)=XI(IL,IK)
220     CONTINUE
        OBJ=OBJFN
        IF (LIST.NE.3)  GO TO 230
        PRINT 540, K
C
230     CALL ZGITLP (N,OLD,XI,T,ALPHA,PT,XINEW,JUMP,NOYES,FF)
C
240     ZGEN=FF
C
C       COMPARE NEW AND OLD POINTS FOR N-DIMENSIONAL MINIMUM
       ACHIEVED
C
        PTDIF=DABS(PT(1)-OLD(1))
        DO 250 IJ=2,N
        ON=DABS (PT(IJ)-OLD(IJ))
        IF (ON.LE.PTDIF) GO TO 250
        PTDIF=ON
250     CONTINUE
        IF (PTDIF.GE.F) GO TO 260
        IF (K.NE.2) GO TO 260
        OBJFN=OBJ
        K=K-1
260     IF (LIST.EQ.0) GO TO 270
        IF (LIST.EQ.2) GO TO 270
        PRINT 430, (IC,R(IC),IC=1,N)
        PRINT 450, ((IC,ID,XJ(IC,ID),IC=1,N),ID=1,N)
        PRINT 550, K
        PRINT 431, (IC,OLD(IC),IC=1,N)
        PRINT 450, ((IC,ID,XI(IC,ID),IC=1,N),ID=1,N)
        PRINT 510, (IC,XINEW(IC),IC=1,N)
        PRINT 520, K,ZGEN
        PRINT 460, (IC,PT(IC),IC=1,N)
        PRINT 420, K,OBJFN
        IF (PTDIF.LT.F) GO TO 280
C
C       TEST FOR N-DIMENSIONAL MINIMUM ACHIEVED
270     IF (COUNT.LT.N) GO TO 320
C       N-DIMENSIONAL MINIMUM ACHIEVED
C
280     CONTINUE
        IF (IBMCRD.LE.1) GO TO 290
        PUNCH 610, IP,IX
        PUNCH 370, N,O,F
        PUNCH 340, (PT(IC),IC=1,N)
        PUNCH 410, LIST,IBMCRD,KLOCK
        IF (IX.EQ.0) GO TO 290
        PUNCH 3,0, ((XI(IC,ID),IC=1,N),ID=1,N)
290     CONTINUE
```

```
      MINFN= FF
300   IF (LIST.LE.1) GO TO 310
      PRINT 350, MINFN
      PRINT 360, (IC,PT(IC),IC=1,N)
      PRINT 440, OBJFN,K
C     TEST FOR FINISH AND RESTART IF NOT
310   IF(IP.GE.ISTOP) GO TO 330
      Q=E
      E=E*E
      K=K+1
      JUMP=JUMP+1
      IP=IP+1
      IPOWEL=IPOWEL*IPOWEL
      GO TO 10
C     REITERATE UNTIL REQUIRED ACCURACY ACHIEVED
320   K=K+1
      JUMP=JUMP+1
      GO TO 190
330   DO 335 J=1,N
335   XOPT(J)=OLD(J)
      PRINT 620
      RETURN
C
340   FORMAT (4(F15.10,5X))
350   FORMAT (1H0,33HOBJECTIVE FUNCTION MINIMUM VALUE=,E18.11)
360   FORMAT (1HC,21HNO.      MINIMUM POINT/(I3,3X,E18.11))
370   FORMAT (I2,3X,F15.10,3X,F15.10)
390   FORMAT (1H1.18HNO. OF DIM OR VAR=,I2,5X,16HONE DIM START
     Q=,E18.11 1,5X,9HACCUPACY=,E18.11)
400   FORMAT (1HC,21HNO.      INITIAL POINT/(I3,3X,E18.11))
410   FORMAT(2I5)
420   FORMAT (1HC,21HITERATIONS 1 THROUGH ,I9,10H REQUIRED ,
     I9,31H OBJEC  1TIVE FUNCTION EVALUATIONS)
430   FORMAT(28H ENTER ZGITER WITH THE POINT,/4H NO.,/(I3,3X,E18.11))
431   FORMAT(26H LEAVE ZGITER AT THE POINT,/4H NO./(I3,3X,E18.11))
440   FORMAT (1HC,32HACHIEVING THIS MINIMUM REQUIRED ,I9,36H
     OBJECTIVE F  1UNCTION EVALUATIONS AND ,I9,11H ITERATIONS)
450   FORMAT (1HC,31H  NO.     XI NO.        DIRECTION/(2X,I3,5X,
     I3,4X,E18 1.11))
460   FORMAT (1HC,17HNO.      NEW POINT/(I3,3X,E18.11))
480   FORMAT (1H0,11HPRINT CODE=,I3,5X,11HPUNCH CODE=,I3,5X,12HTIMING
     CO1DE=,I3,5X,8HXI FLAG=,I3)
510   FORMAT (1HC,24HNO.        XI(N+1)        /(I3,3X,E18.11))
520   FORMAT (1H0,10HITERATION ,I9,37H CHANGES OBJECTIVE FUNCTION VALUE
     1TO ,E18.11)
530   FORMAT (1HC,42HOBJECTIVE FUNCTION VALUE AT INITIAL POINT=,E18.11)
540   FORMAT (1H0,23HENTER ZGITER ITERATION ,I9)
550   FORMAT (1HC,23HLEAVE ZGITER ITERATION ,I9)
560   FORMAT (1HC,32HNO.  INITIALIZATION ENTER POWELL/(I3,3X,E18.11))
580   FORMAT (1HC,17HNO.  ENTER POWELL/(I3,3X,E18.11))
590   FORMAT (1HC,17HNO.  LEAVE POWELL/(I3,3X,E18.11))
600   FORMAT (1HC,32HNO.  INITIALIZATION LEAVE POWELL/(I3,3X,E18.11))
610   FORMAT (2I10)
620   FORMAT (1H1)
      END
```

```
      SUBROUTINE ZGITER (NDV,DPT,DIR,NC,DIST,G,EXTRA,JSW,ISW,FF)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION DPT(20), DIR(20,21), C(20), G(20), H(20), NEW(20), EXTRA
     1(20)
      REAL*8 L,NEW
      INTEGER OBJFN, COUNT
      COMMON/ZANG/EPSILN,O,E,IPOWEL,OBJFN,LIST,COUNT
C
C     UNCONSTRAINED NDV-DIMENSIONAL MINIMIZATION WITHOUT USING DERIVATIV
C        USING GIVEN POINT DPT AND GIVEN DIRECTION DIR
C     NDV=QUANTITY OF VARIABLES IN OBJECTIVE FUNCTION
C     NC=NUMBER OF CURRENT COORDINATE DIRECTION
C     DPT(IC)=IC-TH COMPONENT OF OLD POINT
C     NEW(IC)=IC-TH COMPONENT OF NEW POINT
C     DIR(IC,ID)=IC-TH COMPONENT OF ID-TH NONCOORDINATE NORMALIZED DIREC
C     EXTRA(IC)=IC-TH COMPONENT OF (N+1)-ST ('EXTRA')
C        NORMALIZED NONCOORDINATE DIRECTION
C     DIST=MINIMUM STEP LENGTH ALONG COORDINATE DIRECTION NC
C     C(IC)=IC-TH COMPONENT OF CURRENT NORMALIZED COORDINATE DIRECTION
C     G(IC)=IC-TH COMPONENT OF MINIMUM POINT IN NDV-MINIMIZATION
C     L=MINIMUM STEP LENGTH IN ONE DIMENSIONAL MINIMIZATION ALONG
C        NONCOORDINATE DIRECTION H
C     H(IC)=IC-TH COMPONENT OF CURRENT NONCOORDINATE DIRECTION IN ONE
C        DIMENSIONAL MINIMIZATION
C     COUNT=CURRENT TOTAL OF COORDINATE DIRECTIONS USED WITHOUT COMPUTIN
C        'EXTRA' NONCOORDINATE DIRECTION
C     JSW=FIRST ZGITER ITERATION FLAG
C
C     PART ONE
      COUNT = 0
C     IF K=1 AND IX (IN ZANGWILL) =0, GO TO PART TWO
      IF (JSW.GT.1) GO TO 20
      NC=NC+1
      DO 10 JK=1,NDV
10    NEW(JK)=DPT(JK)
      GO TO 110
C
C     COMPUTE CURRENT COORDINATE DIRECTION
C
20    DO 40 JC=1,NDV
      IF (JC.EQ.NC) GO TO 30
      C(JC)=0.0000
      GO TO 40
30    C(JC)=1.0000
40    CONTINUE
C
C     MINIMIZE IN ONE DIMENSION ALONG CURRENT COORDINATE DIRECTION
C
      IF (LIST.NE.4) GO TO 45
      PRINT 250, (JC,DPT(JC),JC=1,NDV)
45    CALL POWELL (NDV,DPT,C,DIST,FF)
C     TEST FOR ALL COORDINATE DIRECTIONS USED
      IF (NC.NE.NDV) GO TO 50
```

```
      NC=1
      GO TO 60
50    NC=NC+1
C     TEST FOR MINIMUM STEP LENGTH ALONG C(NC)

60    IF (DABS (DIST).GE.EPSILN) GO TO 90
      IF (COUNT.GE.NDV) GO TO 70
      COUNT=COUNT+1
      GO TO 20
C     MINIMUM STEP LENGTH ALONG C(NC) FOUND
70    DO 80 JK=1,NDV
80    G(JK)=OPT(JK)
      RETURN
C
C     UPDATE CURRENT POINT
90    DO 100 JC=1,NDV
100   NEW(JC)=OPT(JC)+DIST*C(JC)
      IF (LIST.NE.4) GO TO 110
      PRINT 260, (JC,NEW(JC),JC=1,NDV)
C
C     PART TWO
C
C     MINIMIZE IN ONE DIMENSION USING CURRENT POINT AND CURRENT
C        NONCOORDINATE DIRECTIONS
110   DO 180 JD=1,NDV
      DO 120 JC=1,NDV
120   H(JC)=DIR(JC,JD)
C
C     TEST FOR ZERO DIRECTION
      IZ=0
      DO 130 J=1,NDV
      IF (DABS (H(J)).LE.1.0D-15) GO TO 130
      IZ=1
130   CONTINUE
      IF (IZ.EQ.0) GO TO 180
      IF(LIST.NE.4) GO TO 155
      PRINT 250, (IJ,NEW(IJ),IJ=1,NDV)
155   CALL POWELL (NDV,NEW,H,L,FF)
160   DO 170 KJ=1,NDV
170   NEW(KJ)=NEW(KJ)+L*H(KJ)
      IF (LIST.NE.4) GO TO 180
      PRINT 260, (JI,NEW(JI),JI=1,NDV)
180   CONTINUE
C
C     COMPUTE 'EXTRA' NONCOORDINATE DIRECTION
C
      DENOM=0.0D00
      DO 190 JC=1,NDV
190   DENOM=DENOM+DABS (NEW(JC)-OPT(JC))
      DO 200 JC=1,NDV
      DIR(JC,NDV+1)=(NEW(JC)-OPT(JC))/DENOM
200   EXTRA(JC)=DIR(JC,NDV+1)
```

```
C       MINIMIZE IN ONE DIMENSION ALONG 'EXTRA' NONCOORDINATE DIRECTION
        DO 210 JC=1,NDV
210     H(JC)=DIR(JC,NDV+1)
        IF (LIST.NE.4) GO TO 215
        PRINT 250, (JC,NEW(JC),JC=1,NDV)
215     CALL POWELL (NDV,NEW,H,L,FF)
        DO 220 JC=1,NDV
220     OPT(JC)=NEW(JC)+L*H(JC)
        IF (LIST.NE.4) GO TO 225
        PRINT 260, (JC,OPT(JC),JC=1,NDV)
C       COMPUTE NONCOORDINATE DIRECTIONS FOR NEXT ITERATION
225     DO 240 JD=1,NDV
        DO 230 JC=1,NDV
230     DIR(JC,JD)=DIR(JC,JD+1)


240     CONTINUE
140     DO 150 JK=1,NDV
150     G(JK)=NEW(JK)
        RETURN
C
250     FORMAT (1HC,17HNO.  POWELL ENTER/(I3,3X,E18.11))
260     FORMAT (1HC,17HNO.  POWELL LEAVE/(I3,3X,E18.11))
        END


        SUBROUTINE POWELL (N,PO,S,STEP,FF)
        IMPLICIT REAL*8 (A-H,O-Z)
        DIMENSION S(20), PO(20), Z(20), F(4), V(4)
        INTEGER OBJFN, COUNT
        COMMON/ZANG/EPSILN,C,E,IPOWEL,OBJFN,LIST,COUNT
C       UNCONSTRAINED ONE DIMENSIONAL MINIMIZATION WITHOUT USING DERIVATIV
C
C       FOR FINDING BELOW REASONABLE Q VALUE, SEE : CALAHAN,D.A., COMPUTER
C          AIDED NETWORK DESIGN., MCGRAW-HILL, 1968.
C       PO=INITIAL POINT VECTOR
C       S=GIVEN DIRECTION VECTOR
C       GIVEN PO,S; FIND STEP THAT MINIMIZES OBJECTIVE FUNCTION PO+STEP*S
C       GIVEN PO,S.  FIND STEP THAT MINIMIZES  OBJECTIVE FUNCTION(PO+STEP*
C       N=QUANTITY OF VARIABLES IN OBJECTIVE FUNCTION
C       Q=INITIAL STEP LENGTH ALONG S
C       R=GEOMETRIC SERIES RATIO FOR FINDING REASONABLE STEP LENGTH IN
C          COMPUTING SET OF INITIAL THREE POINTS
C       CE=GEOMETRIC SERIES COEFFICIENT FOR FINDING ABOVE REASONABLE STEP
C       E=REQUIRED ACCURACY OF MINIMUM POINT (EACH COMPONENT)
C       V=ARRAY OF CURRENT POINT VALUES
C       F=ARRAY OF CURRENT POINT OBJECTIVE FUNCTION VALUES
C       OBJFN=QUANTITY OF OBJECTIVE FUNCTION EVALUATIONS
C       NUM=QUANTITY OF QUADRATIC INTERPOLATIONS
        NUM=0
C       COMPUTE THREE STARTING VALUES AND THEIR OBJECTIVE FUNCTION VALUES
C
        R=1.5D00
        A=0.0D00
```

```
      DO 10 I=1,N
10    Z(I)=P0(I)
      FA=FF
      DO 20 I=1,N
20    Z(I)=P0(I)+Q*S(I)
      FQ=FUNC(Z,N)
      OBJFN=OBJFN+1
      IF (FQ.LT.FA) GO TO 50
      DO 30 I=1,N
30    Z(I)=P0(I)-Q*S(I)
      FNQ=FUNC(Z,N)
      OBJFN=OBJFN+1
      IF (FNQ.GE.FA) GO TO 40
      B=-Q
      FB=FNQ
      ISW=0
      GO TO 60
40    V(1)=-Q
      V(2)=0.0000
      V(3)=Q
      F(1)=FNQ
      F(2)=FA
      F(3)=FQ
      GO TO 120
50    B=Q
      FB=FQ
      ISW=1
60    CF=1.0000
      SUM=1.0000
70    CF=CF*R
      SUM=SUM+CF

      IF (ISW.EQ.1) GO TO 80
      C=-Q*SUM
      GO TO 90
80    C=Q*SUM
90    DO 100 I=1,N
100   Z(I)=P0(I)+C*S(I)
      FC=FUNC(Z,N)
      OBJFN=OBJFN+1
      IF (FC.GT.FB) GO TO 110
      A=B
      FA=FB
      B=C
      FB=FC
      GO TO 70
110   V(1)=A
      V(2)=B
      V(3)=C
      F(1)=FA
      F(2)=FB
      F(3)=FC
C
```

```
C       COMPUTE ZERO OF FIRST DERIVATIVE OF APPROXIMATING QUADRATIC THROUG
C       THREE CURRENT POINTS AND ITS OBJECTIVE FUNCTION VALUE
C
120     W1=V(2)-V(3)
        W2=V(2)+V(3)
        W3=V(3)-V(1)
        W4=V(3)+V(1)
        W5=V(1)-V(2)
        W6=V(1)+V(2)
        W7=2.*(W1*F(1)+W3*F(2)+W5*F(3))
        V(4)=(W1*W2*F(1)+W3*W4*F(2)+W5*W6*F(3))/W7
        DO 130 I=1,N
130     Z(I)=PO(I)+V(4)*S(I)
        F(4)=FUNC(7,N)
        OBJFN=OBJFN+1
        IF (V(4).NE.V(2)) GO TO 140
        GO TO 160
C       TEST THREE CURRENT POINTS FOR CLOSENESS TO ZERO OF FIRST DERIVATIV
140     IJ1=1
150     Y1=DABS (V(IJ1)-V(4))
        IF (Y1-E.GE.0.0D00)  GO TO 170
C       ZERO OF FIRST DERIVATIVE IS MINIMUM DISTANCE ALONG S
160     STEP=V(4)
        FF=F(4)
        IF (F(4).LT.F(2)) GO TO 240
        STEP=V(2)
        FF=F(2)
        GO TO 240
170     IF (IJ1-2) 180,190,200
180     IJ1=2
        GO TO 150
190     IJ1=3
        GO TO 150
C
C       SHRINK CLOSED INTERVAL CONTAINING MINIMUM BY DISCARDING BOTH POINT
C          OUTSIDE OF INTERVAL AND ITS OBJECTIVE FUNCTION VALUE AND
C          RELABELLING REMAINING POINTS AND THEIR OBJECTIVE FUNCTION VALUE
C
200     IF (V(4)-V(2).GT.0.0D00)  GO TO 220
        IF (F(4)-F(2).GT.0.0D00)  GO TO 210
        F(3)=F(2)
        V(3)=V(2)
        F(2)=F(4)
        V(2)=V(4)
        IF (NUM.GE.IPOWEL) GO TO 160
        NUM=NUM+1
        GO TO 120
210     F(1)=F(4)
        V(1)=V(4)
        IF (NUM.GE.IPOWEL) GO TO 160
        NUM=NUM+1
        GO TO 120
```

```
220   IF (F(4)-F(2).LT.0.) GO TO 230
      F(3)=F(4)
      V(3)=V(4)
      IF (NUM.GE.IPOWEL) GO TO 160
      NUM=NUM+1
      GO TO 120
230   F(1)=F(2)
      V(1)=V(2)
      F(2)=F(4)
      V(2)=V(4)
      IF (NUM.GE.IPOWEL) GO TO 160
      NUM=NUM+1
      GO TO 120
240   CONTINUE
      RETURN
      END
```

NELC FESDIR

CATALOG IDENTIFICATION:

E4 FESDIR

PROGRAMMER:

Gail Grotke, Decision and Control Technology Division

PURPOSE:

To minimize a function $f(x_1, x_2, \ldots, x_n)$ of n variables whose values are constrained

RESTRICTIONS AND LIMITATIONS:

The function and constraints, and their first partial derivatives, must be continuous.

LANGUAGE:

FORTRAN IV

COMPUTER CONFIGURATIONS:

IBM 360/65

ENTRY POINTS:

FESDIR

SUBPROGRAMS AND WHERE REFERENCED:

Programmer-Supplied Programs

FUNC called by FESDIR, and POWEL, and BNDY
POWEL called by FESDIR
BNDY called by POWEL

Library Subprograms

SQRT called by FESDIR
ABS called by FESDIR, POWEL, and BNDY

DEFINITION OF VARIABLES

XO	Initial feasible point
X	Working point and final minimum
S	Direction vector
GF	Gradient of the function
C	The constraints
GC	Gradients of the constraints
DELG	Normalized gradient of the violated constraint
DELF	Normalized gradient of the function
NUM	Number of iterations through Powel
N	Number of variables
IC	Number of constraints
STEP	Value returned from Powel that gives the minimum in the S direction
IFLAG	Number of violated constraints

FBOUND	Criterion by which it is determined whether function values are converging
GBOUND	Criterion by which it is determined whether direction vector is converging
CODE	Code that determines what will be printed

INPUT FORMAT:

A driver and a function subprogram are needed.
The driver calls FESDIR (F, XO, N, FBOUND, GBOUND).
The parameters N, IC, FBOUND, GBOUND, CODE and the feasible point XO for FESDIR and the parameters E, IPOWEL, and OBJFN for POWEL must be set in the driver. These three parameters for POWEL are in the common block labeled ZANG. IC and CODE are in common with FFSDIR.

The function FUNC has a parameter list: (K, X, C, GF, GC).
If k = 1 when FUNC is called, only the value of the objective function is returned.
If k = 2, the constraints are evaluated and returned in C.
If k = 3, the gradients of the constraint and the gradient of the function are returned in GC and GF, respectively.
If k = 4, only the gradient of the function is returned.

OUTPUT FORMAT:

Prints out according to the Print Code CODE

3	2	1	0	Value of CODE
√				Function value, point, and constraint at each step in POWEL
√	√	√		Function value returned from POWEL, step size, and minimum point
√	√			Gradient and normalized gradients of violated constraints and function, and their sum
√	√	√	√	Initial and final points, minimum, number of iterations through POWEL, and number of function evaluations

ERROR MESSAGES:

None

PROGRAM DESCRIPTION:

Main Program

The driver sets the values of the parameters and calls FESDIR.

Subroutines and Functions

FESDIR – finds the minimum of the function within the constraint set.
POWEL – is a one-dimensional quadratic search.
FUNC – evaluates the objective function, the constraints, and the gradients of each.
BNDY – checks to see if a point chosen by the quadratic search is within the constraint set and records the number of constraints that are violated.

MATHEMATICAL METHOD:

Given an objective function $f(x_1,x_2, \ldots, x_n)$ and a set of constraints $g_i(x_1,x_2, \ldots, x_n) \leq 0$, FESDIR finds x such that $f(x)$ is minimum and each constraint $g_i(x)$ is satisfied; that is, less than zero. To accomplish this, FESDIR uses a method of feasible directions given in the article "Nonlinear Programming with the Aid of a Multiple-Gradient Summation Technique" by Klingman and Himmelblau.[64] Klingman and Himmelblau suggest using a new direction given by (NSD), new successful direction, defined as

$$(NSD) = \sum_{j=1}^{kc} \left\{ \frac{\text{Grad } C_j(x)}{|\text{Grad } C_j(x)|} + \frac{\text{Grad } F(x)}{|\text{Grad } F(x)|} \right\},$$

where KC is the number of constraints violated and Grad is the gradient or first partial derivative. So (NSD) is the sum of the normalized gradients of the violated constraints and the normalized gradient of the function.

FESDIR uses this idea in choosing a new direction. Starting with a feasible point – that is, a point, **X**, that satisfies all the constraints – and with a direction, **S**, FESDIR uses a one-dimensional quadratic search, POWEL, to find a value STEP such that the minimum feasible point along the direction **S** is given by $X_{min} = X + STEP * S$. If any constraints were violated in finding the minimum, a new direction is determined by the negative of the sum of the normalized gradients of violated constraints and the normalized gradient of the function. Otherwise, the direction is the negative of the normalized gradient of function. The point and direction are then used to find a new minimum point and a new direction. This process is continued until it satisfies the convergence criterion; that is, until the function values converge or the direction vector converges.

1. The Two-Variable Problem

Minimize $\quad f(x_1,x_2) = (x_1-2)^2 + (x_2-1)^2$

subject to $\quad g_1 = x_1 + x_2 - 2 \leq 0$

$\quad g_2 = x_1^2 - x_2^2 \leq 0$

Minimum $\quad f(1,1) = 1$

2. The Circle Problem

Minimize $\quad f(x_1,x_2) = \dfrac{-1}{(x_1-1)^2 + x_2^2}$

Subject to $\quad g_1 = -x_1^2 - x_2^2 + 4 \leq 0$

$\quad g_2 = -16 + x_7^2 + x_2^2 \leq 0$

Minimum $\quad f(-2,0) = -1.0$

3. The Three-Variable Problem

Minimize $f(x_1, x_2, x_3) = x_1^3 - 6x_1^2 + 11x_1 + x_3$

subject to

$$g_1 = x_1^2 + x_2^2 - x_3^2 \leqslant 0$$

$$g_2 = -x_1^2 - x_2^2 - x_3^2 + 4 \leqslant 0$$

$$g_3 = -x_1 \leqslant 0$$

$$g_4 = -x_2 \leqslant 0$$

$$g_5 = -x_3 \leqslant 0$$

$$g_6 = x_3 - 5 \leqslant 0$$

Minimum $f(0, \sqrt{2}, \sqrt{2}) = -\sqrt{2}$

4. The Colville Problem[65]

1. The Two-Variable Problem

	Initial Point	Computed Optimum	Objective Function	Number of Iterations
1.	-1.0	0.99999982	1.0000003	32
2.	2.0	0.99999969		

2. The Circle Problem

1.	2.0	-0.199589 E01	-0.988216	191
2.	2.8	0.141888 E00		

3. The Three-Variable Problem

1.	0.378964 E00	0.17968266 E-08	1.5673426	30
2.	0.168076 E01	0.15673425 E01		
3.	0.234720 E01			

4. The Colville Problem

1.	0.78619999 E02	0.78012675 E02	-0.30631831 E05	29
2.	0.33439999 E02	0.33144035 E02		
3.	0.31070000 E02	0.30144462 E02		
4.	0.44180000 E02	0.44999996 E02		
5.	0.35319999 E02	0.36546304 E02		

```
C       FEASIBLE DIRECTIONS
        DIMENSION X(5), XO(5)
        INTEGER OBJFN,CODE
        COMMON/ZANG/  E,IPOWEL,OBJFN
        COMMON/CON/IC,KQ,CODE,ICON(16)
        KQ=6
        CODE=1
        N=2
        IC=2
        XO(1)=-1.0
        XO(2)=2.0
        FBOUND = 1.0E-12
        GBOUND = 1.0E-12
        E=1.0E-6
        IPOWEL=5
        OBJFN=0
        CALL FESDIR (F,XO,X,N,FBOUND,GBOUND)
        STOP
        END

        FUNCTION FUNC (K,X,C,GF,GC)
        DIMENSION C(16),GF(5),GC(16,5),X(5)
        FUNC = 1.0
        TA=X(1)-2.0
        TB=X(2)-1.0
        GO TO (100,200,300,400), K
100     CONTINUE
        FUNC=TA*TA+TB*TB
        RETURN
200     CONTINUE
        C(1)=X(1)+X(2)-2.0
        C(2)=X(1)*X(1)-X(2)
        RETURN
300     CONTINUE
        GC(1,1)=1.0
        GC(1,2)=1.0
        GC(2,1)=2.0*X(1)
        GC(2,2)=-1.0
400     CONTINUE
        GF(1)=2.0*(X(1)-2.0)
        GF(2)=2.0*(X(2)-1.0)
        RETURN
        END
```

```
      SUBROUTINE FESDIR (F,XO,X,N,FBOUND,GBOUND)
C
C
C     REFERENCE.  'NONLINEAR PROGRAMMING WITH THE AID OF A MULTIPLE-
C     GRADIENT SUMMATION TECHNIQUE' BY W.R. KLINGMAN AND D.M. HIMMELBLAU
C     IN THE JOURNAL OF THE ASSOCIATION FOR COMPUTING MACHINERY, VOL.II,
C     NO. 4 (OCTOBER, 1964), PP. 400-415.
C
C     DEFINITION OF THE VARIABLES.
C     XO        INITIAL FEASIBLE POINT
C     X         WORKING POINT AND FINAL MINIMUM
C     S         DIRECTION VECTOR
C     GF        GRADIENT OF THE FUNCTION
C     C         THE CONSTRAINTS
C     GC        GRADIENTS OF THE CONSTRAINTS
C     DELG      NORMALIZED GRADIENT OF THE VIOLATED CONSTRAINT
C     DELF      NORMALIZED GRADIENT OF THE FUNCTION
C     NUM       NUMBER OF ITERATIONS THROUGH POWEL
C     N         NUMBER OF VARIABLES
C     IC        NUMBER OF CONSTRAINTS
C     STEP      VALUE RETURNED FROM POWEL THAT GIVES THE MINIMUM IN THE
C               S DIRECTION
C     IFLAG     NUMBER OF VIOLATED CONSTRAINTS
C     FBOUND    CRITERION TO SEE IF FUNCTION VALUES ARE CONVERGING
C     GBOUND    CRITERION TO SEE IF DIRECTION VECTOR IS CONVERGING
C     CODE      CODE THAT DETERMINES WHAT WILL BE PRINTED
C
      DIMENSION S(5),GF(5),C(16),GC(16,5),XO(5),X(5),GSUM(5),P(5)
      DIMENSION DELG(5),DELF(5)
      INTEGER OBJFN,CODE
      COMMON/ZANG/  F,IPOWEL,OBJFN
      COMMON/CON/IC,KQ,CODE,ICON(16)
      NUM=0
C
C        START WITH FEASIBLE POINT
      DO 10  J=1,N
10    X(J)=XO(J)
15    VAL = FUNC (4,X,C,GF,GC)
      F = FUNC (1,X,C,GF,GC)
C
C        SET DIRECTION TO THE NORMALIZED GRADIENT OF F
      SUM=0.
      DO 20  J=1,N
      FSQD=GF(J)*GF(J)
20    SUM=SUM+FSQD
      DO 21  J=1,N
21    S(J)=-GF(J)/SQRT(SUM)
C
C        COMPUTE MINIMUM ALONG THE S DIRECTION
25    NUM=NUM+1
      CALL POWEL (N,X,S,STEP,F,IFLAG)
      DO 26  J=1,N
26    X(J)=X(J)+STEP*S(J)
      IF (CODE.NE.0) WRITE(KQ,500) NUM,F,STEP,(J,X(J),J=1,N)
```

```
27    IF (NUM.EQ.1)  GO TO 28
C
C        CHECK FOR CONVERGENCE
      IF ( ABS(F-FSAVE).LT.FBOUND)  GO TO 110
28    FSAVE=F
      IF (IFLAG.EQ.0)  GO TO 15
      VAL = FUNC (3,X,C,GF,GC)
C
C        COMPUTE SUM OF NORMALIZED GRADIENTS OF VIOLATED CONSTRAINTS
      DO 30  J=1,N
30    GSUM(J)=0.0
      DO 60  I=1,IC
      IF (ICON(I).EQ.1)  GO TO 60
      SUM=0.0
      DO 40  J=1,N
      GSQD=GC(I,J)*GC(I,J)
40    SUM=SUM+GSQD
      DO 50  J=1,N
      DELG(J)= GC(I,J)/ SQRT(SUM)
      IF ((GC(I,J).LT.0.).AND.(DELG(J).GT.0.))  DELG(J)=-DELG(J)
50    GSUM(J)= GSUM(J)+DELG(J)
      IF (CODE.LE.1)  GO TO 60
      WRITE (KQ,540) I
      WRITE (KQ,550) (GC(I,J),DELG(J),J=1,N)
60    CONTINUE
C
C        COMPUTE THE NORMALIZED GRADIENT OF F
      SUM=0.0
      DO 70  J=1,N
      FSQD=GF(J)*GF(J)
70    SUM=SUM+FSQD
      ABDELF=  SQRT(SUM)
      DO 80  J=1,N
      DELF(J)= GF(J)/ABDELF
C
C        S IS THE NEW DIRECTION
      S(J)= -DELF(J) -GSUM(J)
80    CONTINUE
      IF (CODE.LE.1)  GO TO 85
      WRITE (KQ,560)
      WRITE (KQ,570) (GF(J),DELF(J),S(J),J=1,N)
85    SUM=0.0
      DO 90  J=1,N
      SSQD=S(J)*S(J)
90    SUM=SUM+SSQD
      DELS=SQRT(SUM)
C
C        CHECK FOR CONVERGENCE
      IF(DELS.LT.GBOUND )  GO TO 110
      DO 100  J=1,N
C
C        S NORMALIZED
100   S(J)=S(J)/DELS
```

```
      GO TO 25
110   WRITE (KQ,510)
      DO 120  J=1,N
120   WRITE (KQ,520) J,XO(J),J,X(J)
      WRITE (KQ,530) F,NUM,OBJFN
      RETURN
500   FORMAT (1H0,I6,21H RETURNED FROM POWELL,/,17H FUNCTION VALUE =,
     1E15.8,/,8H STEP = ,E15.8,/,(3H X(,I2,3H) =,E15.8))
510   FORMAT (1H0,    13HINITIAL POINT,15X,11HFINAL POINT,//)
520   FORMAT (4H XO(,I2,3H) =,E15.8,5X,2HX(,I2,4H) = ,E15.8,/)
530   FORMAT (54H THE MINIMUM FUNCTION VALUE WITHIN THE CONSTRAINTS IS ,
     1E15.8,/,40H THE NUMBER OF ITERATIONS OF POWELL IS =I6,/,
     139H THE NUMBER OF FUNCTION EVALUATIONS IS ,I6)
540   FORMAT (11H CONSTRAINT,I2,/,9H GRADIENT,17X,10HNORMALIZED)
550   FORMAT (1X,E15.8,10X,E15.8)
560   FORMAT (9H FUNCTION,/,9H GRADIENT,17X,10HNORMALIZED,16X,3HSUM)
570   FORMAT (1X,E15.8,10X,E15.8,10X,E15.8)
      END

      SUBROUTINE POWEL  (N,PO,S,STEP,FF,IFLAG)
      DIMENSION S(20), PO(20), Z(20), F(4), V(4), CO(30)
      INTEGER OBJFN
      COMMON/ZANG/  F,IPOWEL,OBJFN
      STEP=0.0
      NUM=0
      R=1.5
      A=0.0
      Q=.1
      GO TO 5
4     Q=Q*.1
      R=1.
      DO 8  I=1,N
8     PO(I)=PO(I)+STEP*S(I)
5     CONTINUE
      FA=FF
      DO 20 I=1,N
20    Z(I)=PO(I)+Q*S(I)
      CALL BNDY (CO,IFLAG,Z,N)
      IF (IFLAG.GE.1)  GO TO 25
      FQ  = FUNC (1,Z,D,GF,GC)
      OBJFN=OBJFN+1
      IF (FQ.LT.FA) GO TO 50
25    CONTINUE
      IFLAGI=IFLAG
      DO 30 I=1,N
30    Z(I)=PO(I)-Q*S(I)
      CALL BNDY (CO,IFLAG,Z,N)
      IF (IFLAG.EQ.0)  GO TO 32
      IF (STEP.EQ.0.)  GO TO 4
      RETURN
32    FNQ = FUNC (1,Z,D,GF,GC)
      OBJFN=OBJFN+1
```

```
      IF(FNQ.GE.FA.AND.IFLAGI.EQ.0)  GO TO 40
      IF (FNQ.LT.FA)  GO TO 35
      IF (STEP.EQ.0.)  GO TO 4
      DO 33  I=1,N
33    Z(I) = PO(I) + Q*S(I)
      CALL BNDY (CO,IFLAG,Z,N)
      RETURN
35    B=-Q
      FB=FNQ
      ISW=0
      GO TO 60
40    STEP=-Q
      FF=FNQ
      RETURN
50    B=Q
      FB=F0
      ISW=1
60    CF=1.0
      SUM=1.0
70    CF=CF*R
      SUM=SUM+CF
      IF (ISW.EQ.1) GO TO 80
      C=-Q*SUM
      GO TO 90
80    C=Q*SUM
90    DO 100 I=1,N
100   Z(I)=PO(I)+C*S(I)
      CALL BNDY (CO,IFLAG,Z,N)
      IF (IFLAG.EQ.0)  GO TO 102
101   STEP=B
      FF=FB
      RETURN
102   CONTINUE
      FC   = FUNC (1,Z,0,GF,GC)
      OBJFN=OBJFN+1
      IF (FC.GT.FB) GO TO 110
      A=B
      FA=FB
      B=C
      FB=FC
      GO TO 70
110   STEP=A
      FF=FA
      IF (STEP.EQ.0.0)  GO TO 4
      RETURN
      END

      SUBROUTINE BNDY (C,IFLAG,X,N)
      DIMENSION C(16),X(5)
      INTEGER CODE
      COMMON/CON/IC,KQ,CODE,ICON(16)
      IFLAG=0
```

```
      VAL = FUNC (2,X,C,GF,GC)
      F = FUNC (1,X,C,GF,GC)
      IF (CODE.NE.3)  GO TO 10
      WRITE (KQ,50) F
      WRITE (KQ,60) (K,X(K),K=1,N)
      WRITE (KQ,70)  (L,C(L),L=1,IC)
10    DO 40  J=1,IC
      IF ( ABS(C(J)).LT.1.0E-15)  C(J)=0.
      IF (C(J).LE.0.0    )  GO TO 30
      ICON(J)=-1
      IFLAG=IFLAG+1
      IF (CODE.NE.3)  GO TO 40
      WRITE (KQ,80)  J
      GO TO 40
30    ICON(J)=1
40    CONTINUE
50    FORMAT (4H F =,F15.8)
60    FORMAT (3H X(,I2,3H) =,E15.8)
70    FORMAT (2H C,I2,E15.8)
80    FORMAT (11H CONSTRAINT,I2,12H IS VIOLATED)
      RETURN
      END
```

RICOCHET GRADIENT (28 SUBROUTINES AND DRIVER)

PROGRAMMER:

J. Greenstadt and R. T. Mertz, IBM/Adapted for Use at NELC by D. C. McCall, Decision and Control Technology Division

PURPOSE:

To find the point $(x_1, x_2, \ldots, x_n)$ at which the objective function $f(x_1, x_2, \ldots, x_n)$ takes on its maximum value, subject to the constraints $g_k(x_1, x_2, \ldots, x_n) \geq 0$

RESTRICTIONS AND LIMITATIONS:

The program handles up to 50 constraints and 50 variables. The first partial derivatives of the function and all constraints must be obtainable.

LANGUAGE:

FORTRAN IV

COMPUTER CONFIGURATIONS:

IBM 360/65

ENTRY POINTS:

Main

SUBPROGRAMS AND WHERE REFERENCED:

Programmer-supplied Programs

PROB called by OBFUNC, OBGRAD, CONSTR, CSTRNM

Library Subprograms

ABS

SQRT

DEFINITION OF VARIABLES:

The manual gives a complete definition of the variables and gives a summary for each subroutine.

INPUT FORMAT:

A subroutine PROB and a set of data cards are needed.

The subroutine defines the objective function, constraints, and gradients. The form used is

SUBROUTINE PROB (NUMX, X, KK, INDX, VAR. GRAD, N, NC, C)

The necessary dimensioning is
DIMENSION X(50), GRAD(50), NC(200), C(200)

DEFINITION OF THE VARIABLES:

NUMX	Serial number of X (unused by PROB)
X	The coordinates of the point
KK = 0	Gives a value of the objective function
1	Gives a value of the KKth constraint
INDX = 0	Gives a function value
1	Gives a gradient value
VAR	Returns the objective or constraint function value
GRAD	Returns the N-gradients
N	Number of variables
NC	Can be used to read in integer data (up to 200)
C	Can be used to read in real data (up to 200)

See sample subroutine.

SAMPLE SUBROUTINE

```
      SUBROUTINE PROB (NUMX,X,KK,INDX,VAR,GRAD,NC,C)
*C    NLO PROBLEM 3
*C    KLINGMAN AND HIMMELBLAU TEST EXAMPLE
C.....
C
      DIMENSION X(50),GRAD(50),NC(200),C(200)
C
C.....
C     ROUTING
100   IF (INDX) 210, 110, 120
C
C     EVALUE]
110   VAL=0.
111   IF(KK) 200,1000, 112
*112  IF(KK-2) 113,1002, 200          These must agree with the no. of constraints,
113   JGO1=KK
*114  GO TO (1001,1002),JGO1          one label for each constraint.
C
C     EGRADIENT]
*120  DO 121 J=1,3                    This must agree with the no. of variables.
121   GRAD(J)=0.
123   IF(KK) 210,2000, 124
*124  IF(KK-2) 125,2020, 210          These must agree with the no. of constraints,
125   JGO2=KK
*126  GO TO (2010,2020),JGO2          one label for each set of constraint gradients.
C
200   VAR=VAL
210   RETURN
C.....
C
C     CALCULATION - VALUES
C
*1000 VAL= 1./((X(1)+1.)*(X(1)+1.)+X(2)*X(2))      Objective function
      GO TO 200
*1001 VAL= X(1)*X(1)+X(2)*X(2)-4.
      GO TO 200                                    Constraints
*1002 VAL= 16.-X(1)*X(1)-X(2)*X(2)
      GO TO 200
C.....
C
C     CALCULATION - GRADIENTS
C
*2000 DENO4=(X(1)+1.)*(X(1)+1.)+X(2)*X(2)
      GRAD(1)=-2.*(X(1)+1.)/DENO4*DENO4            Gradient of objective function
      GRAD(2)=-2.*X(2)/DENO4*DENO4
      GO TO 210
*2010 GRAD(1)= 2.*X(1)
      GRAD(2)= 2.*X(2)                             Gradients of constraints
      GO TO 210
*2020 GRAD(1)=-2.*X(1)
      GRAD(2)=-2.*X(2)
      GO TO 210
C.....
C
      END
```

* Indicates cards that are changed from program to program.

The user-supplied data follow the required data. The user's data are divided into seven classes; preceding each class is a header card with the class number in the first column. All other data cards must be blank in the first column and are read only through column 72.

The seven classes are:

0 Description of the problem

1 Setting for the output control switches (See OUTPUT FORMAT)

2 Integer parameters used in the algorithm subroutines

3 Real parameters used in the algorithm subroutines, including the feasible starting point

Classes 2 and 3 usually use the same parameters from problem to problem except for the starting point.

The starting point begins in the 47th entry under Class 3, and is read in with an E14.8, 3E15.8 format.

4 Size of the problem. The number of variables in columns 2-5; the number of constraints in columns 6-10.

5 Integer Parameters for the PROB subroutine – I4, 15I5 format

6 Real Parameters for the PROB subroutine – E14.8, 3E15.8 format

The reading in of data is terminated by two cards with a 9 in column 1. If the program is to be run more than once with different data, one card with a 9 in column 1 is placed between the data for the different problems and two 9-cards are used to terminate the program.

SAMPLE DATA DECK

```
   NLO CONSTRAINED TEST EXAMPLE FROM KLINGMAN AND HIMMELBLAU PAPER
1
    1     1    11                 1  1     11  1  1  1
   21                           1  111         11
   31                      11       1
   41                           111         1                   Condition codes
   51                                       11                  for printout
   61                   1        11    11                       control
   71                         1                   11         11
   81                          11                  11
   91
  101
2
   20    6   20   20   20   50    6   20   20                   Integer parameters
3
         1.E-15          1.E-4          5.E-4          1.E-3    Real parameters
          1.E-3             4.          1.E-4          1.E-3
          1.E-3          1.E-4          1.E-4          1.E-4
           .999          1.E-3          1.E-3          1.E-3
          5.E-3          1.E-3          5.E-4          1.E-3
          1.E-3             4.          1.E-4          1.E-3
             0.          1.E-4             4.          1.E-3
           100.          1.E-5          1.E-4

  1.                     2.E-2
  2.0              -2.8                                         Initial point
'
    2    3                                                      Program size
6
9
9
/*
```

See pages 1[illegible]-16 for further information.

OUTPUT FORMAT:

The output for the program is controlled by control switches set under the first class of data. Following the header card are 10 cards numbered consecutively from 1-10 in columns 4 and 5. The switch controls are in columns 6 through 72. A "1" punch indicates the value will be printed; a blank indicates the value will be omitted. In the description of each subroutine in the manual the code for printing the values is given. For example, to print the final function value from the subroutine MONITR, we use the code 1, 44, F (on page 38). A "1' in column 44 of the first card will cause F to be printed. This code appears with the printout to help with the identification of the values.

A good choice for the minimal amount of printout (just the final result) is 1's in columns 11, 16, 17, 35, 38, 44, 45, 48, 51, 54 on the first card; the other cards are blank in columns 6-72.

The maximum printout occurs when all 10 cards are blank in columns 6 through 72. For debugging purposes the values desired may be chosen from the manual.

MATHEMATICAL METHOD AND REFERENCE:

See the manual Contributed Program Library #360D-15.3001 'Non-Linear Optimization-Ricochet Gradient Method.'[66]

TEST EXAMPLES:

The following problems were tested with this program:

Problem 1.

Minimize $f = (x_1 - 2)^2 + (x_2 - 1)^2$

$$g_1 = x_1 + x_2 - 2 \leqslant 0$$

$$g_2 = x_1^2 - x_2 \leqslant 0$$

$$f_{min.} = 1.0$$

Problem 2. (Circle)

Maximize $$f = \frac{1}{(x_1 + 1)^2 + x_2^2}$$

subject to $$g_1 = x_1^2 + x_2^2 - 4 \geqslant 0$$

$$g_2 = 16 - x_1^2 - x_2^2 \geqslant 0$$

$$f_{max.} = 1.0$$

Ref. Klingman and Himmelblau[65]

Problem 3.

Maximize $f = y + \sin x$

subject to $0 \leqslant x \leqslant 1$

$0 \leqslant y \leqslant 1$

$x^2 + y^2 \leqslant 1$

$f_{max.} = 1.366$

Ref. problems 3 through 6 were from Krolak and Cooper as found in the Klingman and Himmelblau paper.[64]

Problem 4.

Maximize $f = -(y - x)^4 + (1 - x)$

subject to $0.2 \leqslant x \leqslant 2.0$

$0.2 \leqslant y \leqslant 2.0$

$x^2 + y^2 \leqslant 1$

$f_{max.} = 0.8$

Problem 5.

Maximize $f = -x^2 + x - y^2 + y + 4$

subject to $0.2 \leqslant x \leqslant 2.0$

$0.2 \leqslant y \leqslant 2.0$

$x^2 + y^2 \leqslant 4$

$f_{max.} = 4.5$

Problem 6.

Maximize $f = \exp\left(-(x - 1)^2 - \dfrac{(y^2 - 0.5)^2}{0.132}\right)$

subject to $0.2 \leqslant x \leqslant 2.0$

$0.2 \leqslant y \leqslant 2.0$

$x^2 + y^2 \leqslant 4$

$f_{max.} = 1.0$

Problem 7. (Lootsma)

Minimize $f = x_1^3 - 6x_1^2 + 11x_1 + x_3$

subject to $-x_1^2 - x_2^2 + x_3^2 \geqslant 0$ $\quad x_1 \geqslant 0$

$x_1^2 - x_2^2 + x_3^2 - 4 \geqslant 0$ $\quad x_2 \geqslant 0$

$-x_3 + 5 \geqslant 0$ $\quad x_3 \geqslant 0$

$f_{min.} = \sqrt{2}$

Ref. Lootsma, F. A.[46]

Problem 8.

Minimize $\sum_{j=1}^{5} e_j y_j + \sum_{j=1}^{5} \sum_{i=1}^{5} c_{ij} y_i y_j + \sum_{j=1}^{5} d_j y_j^3$

subject to $\sum_{j=1}^{5} a_{ij} \geqslant b_i$, where i = 1, . . ., 10

$f_{min.} = -32.349$

i \ j	1	2	3	4	5	b_i
a_{ij} 1	-16	2	0	1	0	-40
2	0	-2	0	0.4	2	-2
3	-3.5	0	2	0	0	-2.5
4	0	-2	0	-4	-1	-4
5	0	-9	-2	1	-2.8	-4
6	2	0	-4	0	0	-1
7	-1	-1	-1	-1	-1	-40
8	-1	-2	-3	-2	-1	-60
9	1	2	3	4	5	5
10	1	1	1	1	1	1
c_{ij} 1	30	-20	-10	32	-10	
2	-20	39	-6	-31	32	
3	-10	-6	10	-6	-10	
4	32	-31	-6	39	-20	
5	-10	32	-10	-20	30	
d_j	4	8	10	6	2	
e_j	-15	-27	-36	-18	-12	

Ref. Colville, A. R.[65]

RESULTS OF TESTED EXAMPLES:

1. Problem 1

Initial Point	Computed Optimum	Objective Function	Number of Iterations
x_1 -1.0	0.99999994	-1.000	47
x_2 2.0	0.99999988		
x_1 -1.0	0.99999994	-1.00	36
x_2 0.0	0.99999988		
x_1 2.0	0.99999994	-1.000	7
x_2 2.0	0.99999988		

2. Circle

x_1 2.00	-2.0	1.000	265
x_2 -2.8	0.00076647		
x_1 3.0	2.0	0.111111	12
x_2 0.0	0.0		
x_1 5.0	-1.9999857	0.999971	304
x_2 5.0	-0.00756613		
x_1 0.0	-1.99999914	0.999982	196
x_2 3.0	0.0060059		
x_1 1.0	-2.0	1.000	316
x_2 1.0	-0.000674		

3. Krolak and Cooper #1

Initial Point		Computed Optimum	Objective Function	Number of Iterations
x_1	0.0	0.629556	1.3657408	33
x_2	0.9	0.776955		
x_1	-0.3	0.63038069	1.3657379	29
x_2	-0.3	0.77628624		
x_1	0.0	0.63037407	1.3657379	30
x_2	-2.0	0.77629149		
x_1	1.0	0.62933254	1.3657408	53
x_2	-2.0	0.77713621		
x_1	1.0	0.63037902	1.3657379	30
x_2	1.0	0.77628750		

4. Krolak and Cooper #2

x_1	-.5	0.19999999	0.80000001	7
x_2	-.5	0.19999999		
x	2.0	0.19999999	0.80000001	25
x_2	0.0	0.19999999		
x_1	1.0	0.19999999	0.80000001	29
x_2	1.0	0.19999999		

5. Krolak and Cooper #3

Initial Point		Computed Optimum	Objective Function	Number of Iterations
x_1	0.0	0.50000012	4.5000	36
x_2	0.9	0.50000018		
x_1	-0.3	0.50000030	4.5000	36
x_2	-0.3	0.50000030		
x_1	0.0	0.50000006	4.5000	38
x_2	-2.0	0.50000006		
x_1	1.0	0.50000000	4.5000	6
x_2	0.0	0.49999994		
x_1	1.0	0.50000006	4.5000	37
x_2	1.0	0.50000006		

6. Krolak and Cooper #4

x_1	-0.5	0.99974561	0.99999994	106
x_2	-0.5	0.70710695		
x_1	2.0	1.0002337	0.99999994	82
x_2	0.0	0.70710659		
x_1	1.0	1.0000000	1.000000	13
x_2	1.0	0.70711035		

7. Lootsma

x_1	0.37896395	0.0		
x_2	1.6807594	1.4142141	1.4142141	52
x_3	2.3471994	1.4142141		

8. Colville 1

Initial Point		Computed Optimum	Objective Function	Number of Iterations
x_1	0.0	0.30000037	-32.343491	337
x_2	0.0	0.33695012		
x_3	0.0	0.40000027		
x_4	0.0	0.43677974		
x_5	1.0	0.21579641		

APPENDIX 3: REFERENCES

1. Dejka, W. J. and McCall, D. C., "A Study in the Design of a Practical Tunable Bandpass Filter Using Mathematical Programming," p. 267-270 in IEEE Systems Science and Cybernetics Conference, Pittsburgh, 1970, Systems For the Seventies; Proceedings [Held] October 14-16, 1970, at Pittsburgh, Pennsylvania, Institute of Electrical and Electronic Engineers, 1970

2. Lasdon, L. S. and Waren, A. D., "Mathematical Programming For Optimal Design," Electro-Technology, v. 80, p. 53-70, November 1967

3. Calahan, D. A., Computer-Aided Network Design, McGraw-Hill, 1968

4. "Special Issue on Computer-Aided Design," Institute of Electrical and Electronics Engineers. Proceedings, v. 55, No. 11, November 1967

5. Joint Conference on Mathematical and Computer Aids to Design, Anaheim, California, 1969, Digest Record, Institute of Electrical and Electronics Engineers, 1969 (IEEE Catalogue No. 69C63-C)

6. Rosen, J. B. and Meyer, R., "Solution of Nonlinear Two-Point Bounding Value Problems by Linear Programming," p. 71-84 in Conference on the Mathematical Theory of Control, University of Southern California, 1967, Mathematical Theory of Control: Proceedings of a Conference Held at the University of Southern California, Los Angeles, January 30 – February 1, 1967, Academic Press, 1967

7. Naval Electronics Laboratory Center Technical Document 82, Direct-Search Methods For the Solution of the Two-Point Boundary Value Problem [TPBVP], by W. J. Dejka, 30 January 1970

8. Naval Electronics Laboratory Center Technical Note 1417,* Computer Programs to Find the Minimum of a Nonlinear Function of Several Variables, by P. S. Winterbauer, 1 August 1968

9. Fletcher, R. and Powell, M. J. D., "A Rapidly Convergent Descent Method For Minimization," Computer Journal, v. 6, p. 163-168, 1963

*NELC technical notes are informal documents intended chiefly for use within the Center.

10. Fletcher, R. and Reeves, C. M., "Function Minimization by Conjugate Gradients," Computer Journal, v. 7, p. 149-154, 1964

11. IBM Application Program H2-0205-3 System/360 Scientific Subroutine Package (360A-CM-03X) Version III: Programmer's Manual, 1968

12. Naval Electronics Laboratory Center Technical Note 1628, Two Direct Search Methods of Mathematical Programming, by D. C. McCall and C. T. Ogata, 26 January 1970

13. Zangwill, W. I., "Minimizing a Function Without Calculating Derivatives," Computer Journal, v. 10, p. 293-296, 1967

14. Hooke, R. and Jeeves, T. A., " 'Direct Search' Solution of Numerical and Statistical Problems," Association For Computing Machinery. Journal, v. 8, p. 212-229, 1961

15. Leon, A., "A Comparison Among Eight Known Optimization Procedures," p. 23-46 in Symposium on Recent Advances in Optimization Techniques, Carnegie Institute of Technology, 1965, Edited by T. P. Vogl, Wiley, 1966

16. Hadley, G., Linear Programming, Addison-Wesley, 1962

17. Zoutendijk, G., Methods of Feasible Directions, Elsevier Publishing Company, 1960

18. IBM Application Program H20-0476-1, Mathematical Programming System/360 (360A-CO-14X) Version 2, Linear and Separable Programming – User's Manual, 30 April 1969

19. IBM Application Program H20-0603-0, Mathematical Programming System/360 (360A-CO-14X) Message Manual, March 1969

20. IBM Application Program H20-0372-1, Mathematical Programming System/360 (360A-CO-14X) Read Communications Format [READCOMM]: Program Reference Manual, December 1968

21. Griffith, R. E. and Stewart, R. A., "A Nonlinear Programming Technique For the Optimization of Continuous Processing Systems," Management Science, v. 7, p. 379-392, July 1961

22. Rosen, J. B., "The Gradient Projection Method For Nonlinear Programming: Part 1, Linear Constraints," Society For Industrial and Applied Mathematics. Journal, v. 8, p. 181-217, March 1960

23. California Institute of Technology. Jet Propulsion Laboratory Technical Memorandum 240, Computing Quadratic Programming Problems: Linear Inequality Constraints, by R. J. Hanson, 9 February 1970

24. Fiacco, A. V. and McCormick, G. P., Nonlinear Programming: Sequential Unconstrained Minimization Techniques, Wiley, 1968

25. Hartley, H. O. and Hocking, R. R., "Convex Programming by Tangential Approximation," Management Science, v. 9, p. 600-612, July 1963

26. IBM. Contributed Program Library 360D-15.3.001, Non-Linear Optimization – Ricochet Gradient Method, by J. Greenstadt and R. T. Mertz, October 1967

27. Research Analysis Corporation, A User's Manual For Experimental SUMT: The Computer Program Implementing the Sequential Unconstrained Minimization Technique For Nonlinear Programming, by G. P. McCormick and others, March 1970

28. IBM. Program Information Department Program 360D-15.2.011, Zero-One Integer Linear Programming With Heuristics, by B. D. Holcomb, 1968

29. IBM. Program Information Department Program 3600-15.2.005, Branch and Bound Mixed Integer Programming BBMIP, by R. Shareshian, April 1967

30. Pace Company, Comparison of Integer Programming Algorithms, by W. J. Mears and G. S. Dawkins, 1-3 May 1968 (Paper presented at the Joint National Meeting of the Operations Research Society of America and the Institute of Management Sciences, San Francisco)

31. Stanford University. Operations Research Department Technical Report 11, Bound-and-Scan Algorithm For Pure Integer Linear Programming With General Variables, by F. S. Hillier, 20 May 1969

32. Wolfe, P., "Review of Nonlinear Programming," p. xi-xv in Optimization, edited by R. Fletcher, Academic Press, 1969

33. IBM Application Program H20-0147-0, 1620 Electronic Circuit Analysis Program [ECAP] (1620-EE-02X); Application Description, n.d.

34. Lee, H. B. and others, "Program Refines Circuit From Rough Design Data," Electronics, v. 43, p. 58-65, 23 November 1970

35. University of California, Los Angeles. Department of Engineering, A Critical Evaluation of the Numerical Significance of General Circuit Analysis Programs in Relation to the Transient Problem, by D. E. Meyerhoff and L. P. McNamee, n.d.

36. Aoki, M., Introduction to Optimization Techniques Fundamentals and Applications of Nonlinear Programming, Macmillan, 1971

37. Gear, C. W., "The Automatic Integration of Large Systems of Ordinary Differential Equations," p. 27-29 in Joint Conference on Mathematical and Computer Aids to Design, Anaheim, California, 1969, Digest Record, Institute of Electrical and Electronics Engineers, 1969 (IEEE Catalogue No. 69C63-C)

38. Calahan, D. A., "Numerical Considerations in the Transient Analysis and the Optimal Design of Nonlinear Circuits," p. 129-145 in Joint Conference on Mathematical and Computer Aids to Design, Anaheim, California, 1969, Digest Record, Institute of Electrical and Electronics Engineers, 1969 (IEEE Catalogue No. 69C63-C)

39. Box, M. J., "A Comparison of Several Current Optimization Methods and the Use of Transformations in Constrained Problems," Computer Journal, v. 9, p. 67-77, 1966

40 Hildebrand, F. B., Introduction to Numerical Analysis, McGraw-Hill, 1956

41. Stewart, G. W., III, "A Modification of Davidon's Minimization Method to Accept Difference Approximations of Derivatives," Association For Computing Machinery. Journal, v. 14, p. 72-83, January 1967

42. AEC Research Development Report ANL-5990, Revision 2, Variable Metric Method For Minimization, by W. C. Davidon, February 1966

43. Rabinowitz, P., "Applications of Linear Programming to Numerical Analysis," S I A M Review (Society For Industrial and Applied Mathematics), v. 10, p. 128-159, April 1968

44. Zangwill, W. I., Nonlinear Programming: A Unified Approach, Prentice Hall, 1969

45. Kuhn, H. W. and Tucker, A. W., "Nonlinear Programming," p. 481-492 in Berkeley Symposium on Mathematical Statistics and Probability, Proceedings. Second, University of California, 1950, University of California Press, 1951

46. Lootsma, F. A., "Logarithmic Programming: A Method of Solving Nonlinear Programming Problems," Philips Research Reports, v. 22, p. 329-344, June 1967

47. Zoutendijk, G., "Nonlinear Programming: A Numerical Survey," S I A M Journal on Control (Society For Industrial and Applied Mathematics), v. 4, p. 194-210, February 1966

48. Pierre, D. A., Optimization Theory With Applications, Wiley, 1969

49. Gomory, R. E., "Outline of an Algorithm For Integer Solutions to Linear Programs," American Mathematical Society. Bulletin, v. 64, p. 275-278, September 1958

50. Stanford University. Operations Research Department Technical Report 10, Efficient Heuristic Procedures For Integer Linear Programming With an Interior, by F. S. Hillier, 28 February 1969

51. Illinois. University. Computer Science Department Report 264, Logical Design of an Optimal Network by Integer Linear Programming, Part 1, by S. Muroga, 18 July 1968

52. Pyne, I. B. and McCluskey, E. J., "An Essay on Prime Implicant Tables," Society For Industrial and Applied Mathematics. Journal, v. 9, p. 604-631, December 1961

53. Breuer, M. A., "Logic Synthesis," p. 146-164 in Joint Conference on Mathematical and Computer Aids to Design, Anaheim, California, 1969, Digest Record, Institute of Electrical and Electronics Engineers, 1969 (IEEE Catalogue No. 69C63-C)

54. Kodres, U. R., "Logic Circuit Layout," p. 165-191 in Joint Conference on Mathematical and Computer Aids to Design, Anaheim, California, 1969, Digest Record, Institute of Electrical and Electronics Engineers, 1969 (IEEE Catalogue No. 69C63-C)

55. Breuer, M. A., "The Application of Integer Programming in Design Automation," SHARE Design Automation Workshop, Proceedings, 1966

56. McCluskey, E. J., "Error Correcting Codes: A Linear Programming Approach," Bell System Technical Journal, v. 38, p. 1485-1512, November 1959

57. Karp, R. M., "Minimum-Redundancy Coding For the Discrete Noiseless Channel," Institute of Radio Engineers. Transactions: Information Theory, v. IT-7, p. 27-38, January 1961

58. Saaty, T. L. and Suzuki, G., "A Nonlinear Programming Model in Optimum Communication Satellite Use," S I A M Review (Society For Industrial and Applied Mathematics), v. 7, p. 403-408, July 1965

59. Carnegie Institute of Technology ONR Research Memorandum 116, A Study of the All-Integer Integer Programming Algorithm, by F. Glover, September 1963

60. Rosenbrock, H. H., "An Automatic Method For Finding the Greatest or Least Value of a Function," Computer Journal, v. 3, p. 175-184, 1960

61. Powell, M. J. D., "An Efficient Method of Finding the Minimum of a Function of Several Variables Without Calculating Derivatives," Computer Journal, v. 7, p. 155, 1964

62. Powell, M. J. D., "An Iterative Method for Finding Stationary Values of a Function of Several Variables," Computer Journal, v. 5, p. 147, 1962

63. Fletcher, R., "Function Minimization Without Evaluating Derivatives – a Review," Computer Journal, v. 8, p. 33, 1965

64. Klingman, W. R. and Himmelblau, D. M., "Nonlinear Programming With the Aid of a Multiple-Gradient Summation Technique," Association For Computing Machinery. Journal, v. 11, p. 400-415, October 1964

65. IBM Data Processing Division Technical Report No. 320-2949, A Comparative Study on Nonlinear Programming Codes, New York Scientific Center, by A. R. Colville, June 1968

66. Contributed Program Library #360D-15.3001, Non-Linear Optimization – Ricochet Gradient Method